МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Нижегородский государственный университет им. Н.И. Лобачевского

В.Е. Алексеев
Д.В. Захарова

ТЕОРИЯ ГРАФОВ

Учебное пособие

Рекомендовано методической комиссией ИИТММ
для студентов ННГУ, обучающихся по направлению подготовки
02.03.02 « Фундаментальна информатика и информационные технологии»

Нижний Новгород
2017

УДК 519.17
ББК В182
А - 47

А-47 Алексеев В.Е, Захарова Д.В. ТЕОРИЯ ГРАФОВ: Учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. –119 с.

Рецензенты: д. ф.-м. н., проф. М.А. Иорданский,
к. ф.-м. н., доц. В.А. Зорин

В учебном пособии излагаются основные понятия и фундаментальные факты теории графов, методы метрического и структурного анализа графов, алгоритмы решения экстремальных задач на графах. Рассматриваются важнейшие классы графов: деревья, двудольные графы, планарные графы. Пособие содержит также задачи для практических занятий и задания для самостоятельной работы студентов.

Учебное пособие предназначено для студентов ННГУ, обучающихся по направлению подготовки 02.03.02 «Фундаментальная информатика и информационные технологии», изучающих курс «Теория графов».

Ответственный за выпуск:
зам. председателя методической комиссии ИИТММ ННГУ,
к.т.н, доцент **В.М. Сморкалова**

УДК 519.17
ББК В182

Оглавление

Предисловие

Настоящее учебное пособие является переработанным вариантом электронного пособия [1]. В нем добавлены отсутствующие в электронном издании доказательства теорем, за исключением некоторых наиболее сложных. Добавлено также значительное количество задач, некоторые из них могут служить темами курсовых работ – они помечены двумя звездочками.

Пособие предназначено для литературно-методической поддержки курса «Теория графов», содержащийся в нем материал может также использоваться в курсе дискретной математики и в спецкурсах, где затрагиваются вопросы теории графов и разработки алгоритмов на графах.

1. Начальные понятия

1.1. Определение графа. Типы графов

Граф – математический объект, состоящий из двух множеств. Одно из них – любое конечное множество, его элементы называются *вершинами* графа. Другое множество состоит из пар вершин, эти пары называются *ребрами* графа. Если множество вершин графа обозначено буквой V, множество ребер – буквой E, а сам граф – буквой G, то пишут $G = (V, E)$.

Один момент в этом определении требует уточнения: ребра – это упорядоченные пары вершин или неупорядоченные? Иначе говоря, считаем ли мы ребра (a, b) и (b, a) различными, или это одно и то же ребро? Если ребра – упорядоченные пары, то такой граф называется *ориентированным* (сокращенно *орграф*), если же неупорядоченные, то *неориентированным*.

Говорят, что ребро (a, b) *соединяет* вершины a и b. Заметим, что в графе может быть не более одного ребра, соединяющего две данные вершины. Ребро типа (a, a), т.е. соединяющее вершину с ней же самой, называют *петлей*. Иногда петли разрешаются, иногда запрещаются. В последнем случае говорят, что рассматриваются графы без петель.

В дальнейшем, если не оговаривается иное, под графом понимается неориентированный граф без петель, такие графы называют *обыкновенными*.

Для обозначения числа вершин и числа ребер графа будем обычно использовать буквы n и m.

Мультиграф – обобщение понятия графа. В мультиграфе могут быть *кратные ребра*, то есть несколько ребер, соединяющих одну и ту же пару вершин. Иначе говоря, в мультиграфе E является мультимножеством, то есть одна пара может входить в него несколько раз.

1.2. Смежность, инцидентность, степени

Если в графе есть ребро (a, b), то говорят, что вершины a и b в нем *смежны*.

Говорят, что ребро $e = (a, b)$ *инцидентно* каждой из вершин a и b, а каждая из этих вершин *инцидентна* ребру e.

Множество вершин, смежных с данной вершиной x в некотором графе, называется *окрестностью* этой вершины и обозначается через $N(x)$. Число вершин в $N(x)$ называется *степенью* вершины x и обозначается через $\deg(x)$, т.е. $\deg(x) = |N(x)|$.

Если сложить степени всех вершин графа, то каждое ребро внесет в эту сумму вклад, равный 2. Следовательно, сумма степеней всех вершин

равна удвоенному числу ребер графа. Это утверждение называют теоремой о рукопожатиях.

Теорема 1.1 (о рукопожатиях). *Для графа с m ребрами и множеством вершин V выполняется равенство*

$$\sum_{x \in V} \deg(x) = 2m.$$

В ориентированном графе для каждой вершины x можно ввести два числа: $\deg^+(x)$– число входящих ребер, т.е. ребер вида (y, x) и $\deg^-(x)$ – число выходящих, т.е. ребер вида (x, y) . Эти числа называют соответственно *степенью захода* и *степенью исхода* вершины x. Аналогом теоремы о рукопожатиях для ориентированного графа является очевидное равенство

$$\sum_{x \in V} \deg^+(x) = \sum_{x \in V} \deg^-(x).$$

1.3. Способы задания графов

Существует много способов представить граф, назовем только самые распространенные. Все они иллюстрируются на примере одного графа.

1. Перечисление элементов. Исходя из определения, для того, чтобы задать граф, достаточно перечислить его вершины и ребра (т.е. пары вершин). Пример:

   $V = \{a, b, c, d, e\}$,

   $E = \{(a, b), (a, c), (a, e), (b, c), (b, d), (c, e), (d, e)\}$.

2. Изображение. Если граф не слишком большой, его можно нарисовать. В неориентированном графе ребра изображают линиями, соединяющими смежные вершины, в ориентированном – стрелками. На рисунке 1.1 показан граф, заданный выше перечислением вершин и ребер.

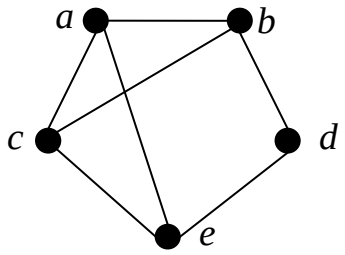


Рис. 1.1. Изображение графа

3. Матрица смежности. Пусть G – граф с n вершинами, пронумерованными числами от 1 до n. Матрица смежности – это таблица с n строками и n столбцами, в которой элемент, стоящий на пересечении строки с номером i и столбца с номером j, равен 1, если вершины с номерами i и j смежны, и 0, если они не смежны.
   Пример (вершины пронумерованы в алфавитном порядке):

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

   На главной диагонали матрицы смежности обыкновенного графа всегда стоят нули (нет петель) и эта матрица симметрична относительно главной диагонали (граф неориентированный).
4. Матрица инцидентности. Пусть G – граф, вершины которого пронумерованы числами от 1 до n, а ребра – числами от 1 до m. В матрице инцидентности строки соответствуют вершинам, а столбцы – ребрам. На пересечении строки с номером i и столбца с номером j стоит 1, если вершина с номером i инцидентна ребру с номером j, и 0 в противном случае.
   Пример:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

5. Списки смежности. Этот способ часто используется для компьютерного представления графов. Состоит он в том, что для каждой вершины задается список всех смежных с ней вершин. В структурах данных, применяемых в программировании, списки смежности могут быть реализованы как массив линейных списков. В формулировках задач будем эти списки оформлять так: пишется номер или имя вершины и после двоеточия перечисляются все смежные с ней вершины.
   Пример:

   1: 2, 3, 5;

   2: 1, 3, 4;

   3: 1, 2, 5;

   4: 2, 5;

   5: 1, 3, 4.

1.4. Некоторые специальные графы

Множество $\{1,2,\dots,n\}$ в дальнейшем будем обозначать V_n.

Пустой граф – граф, не содержащий ни одного ребра. Пустой граф с множеством вершин V_n обозначается O_n.

Полный граф – граф, в котором каждые две вершины смежны. Полный граф с множеством вершин V_n обозначается K_n.

Путь P_n имеет множество вершин V_n, ребрами его являются пары $(i, i+1)$, $i = 1,2,\dots,n-1$.

Цикл C_n получается из графа P_n добавлением ребра $(1, n)$.

Полный двудольный граф $K_{p,q}$. Множество вершин состоит из двух частей, в одной из них p вершин, в другой q вершин. Любые две вершины из одной части не смежны, любые две вершины из разных частей смежны. Полный двудольный граф, в котором одна доля состоит из одной вершины, называют *звездой*.

1.5. Изоморфизм

Графы $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ называются *изоморфными*, если существует такая биекция f множества V_1 на множество V_2, что $(a, b) \in E_1$ тогда и только тогда, когда $(f(a), f(b)) \in E_2$. Отображение f в этом случае называется *изоморфизмом* графа G_1 на граф G_2. Это определение изоморфизма годится и для ориентированных графов, нужно только обе упоминаемые в нем пары вершин считать упорядоченными.

Тот факт, что графы G_1 и G_2 изоморфны, записывается так: $G_1 \cong G_2$.

Изоморфизм – бинарное отношение на множестве графов. Очевидно, это отношение рефлексивно, симметрично и транзитивно, то есть является отношением эквивалентности. Классы эквивалентности называются *абстрактными графами*. Когда говорят, что рассматриваются абстрактные графы, это означает, что изоморфные графы считаются одинаковыми. Абстрактный граф можно представлять себе как граф, у изображения которого стерты названия (пометки) вершин, поэтому абстрактные графы иногда называют также *непомеченными* графами.

В общем случае узнать, изоморфны ли два графа, достаточно сложно. Если буквально следовать определению, то нужно перебрать все биекции множества вершин одного из них на множество вершин другого и для каждой из этих биекций проверить, является ли она изоморфизмом. Для n вершин имеется $n!$ биекций и эта работа становится практически невыполнимой уже для не очень больших n (например, $20! > 2 \cdot 10^{18}$). Однако во многих случаях бывает довольно легко установить, что два данных графа неизоморфны. Так, при изоморфизме смежные вершины переходят в смежные, а несмежные – в несмежные. Значит, у изоморфных графов должно быть одинаковое число ребер. Ясно, что при изоморфизме

каждая вершина одного графа отображается в вершину той же степени другого графа. Значит, если выписать степени вершин графа в порядке, скажем, неубывания, то полученные наборы степеней у изоморфных графов должны совпадать.

Характеристики или свойства графов, которые сохраняются при изоморфизме, называются *инвариантами*. Число ребер и набор степеней – примеры легко вычисляемых инвариантов. Если у двух графов разные наборы степеней, то эти графы гарантированно не изоморфны. К сожалению, обратное неверно – неизоморфные графы могут иметь одинаковые наборы степеней. Пример таких графов показан на рисунке 1.2. Изображенные графы имеют одинаковые наборы степеней (2, 2, 3, 3, 3, 3), но они не изоморфны. Это можно заключить из того, например, что в левом графе имеется *треугольник* – три попарно смежные вершины, а в правом такого нет. Ясно, что при изоморфизме треугольник переходит в треугольник, то есть наличие треугольника в графе – это инвариант

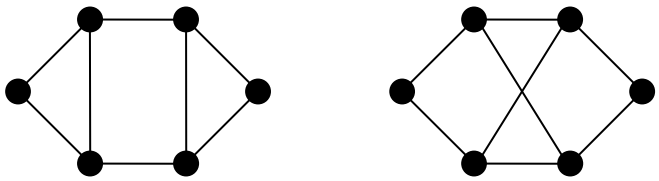


Рис. 1.2. Неизоморфные графы с одинаковыми наборами степеней

Если удается установить, что для двух исследуемых графов некоторый инвариант принимает разные значения, то эти графы неизоморфны. Было бы полезно иметь полную систему инвариантов, то есть такую, чтобы совпадение всех этих инвариантов у двух графов гарантировало изоморфность этих графов. К сожалению, известные на сегодня полные системы инвариантов имеют тот же недостаток, что и непосредственная проверка изоморфности графов перебором биекций – их вычисление требует слишком много времени. Поиски полной системы достаточно быстро вычисляемых инвариантов ведутся давно, но пока не увенчались успехом.

1.6. Подграфы

Граф $G' = (V', E')$ называется *подграфом* графа $G = (V, E)$, если $V' \subseteq V, E' \subseteq E$. Всякий подграф может быть получен из графа удалением некоторых вершин и ребер (при удалении вершины удаляются и все инцидентные ей ребра).

Подграф G' графа G называется *остовным*, если $V' = V$. Остовный подграф получается удалением из графа некоторых ребер, вершины же остаются в неприкосновенности.

Порожденный подграф получается из графа удалением некоторых вершин. Все ребра, которые были в графе между оставшимися вершинами, должны сохраниться в подграфе. Говорят, что подграф порождается оставшимися вершинами.

Допустим, имеется граф G с множеством вершин V_n. Удалив вершину i, получаем порожденный подграф G_i. Таким образом, получим n порожденных подграфов $G_1, G_2, \dots, G_n$, каждый с $n-1$ вершиной. Допустим, что эти n графов нам известны как абстрактные графы, то есть номера, которые имели вершины этих подграфов в графе G, не указаны («стерты»). Спрашивается, можно ли по этим абстрактным графам восстановить граф G (тоже как абстрактный граф)? Очевидно, при $n = 2$ нельзя. Гипотеза, высказанная еще в 1945 г., утверждает, что при $n > 2$ любой граф можно восстановить, то есть набор непомеченных графов $G_1, G_2, \dots, G_n$ определяет граф G с точностью до изоморфизма. Это одна из наиболее известных нерешенных проблем теории графов – проблема восстановления.

1.7. Операции над графами

Для графов $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ их *объединение* $G_1 \cup G_2$ определяется как граф $(V_1 \cup V_2, E_1 \cup E_2)$, а *пересечение* $G_1 \cap G_2$ – как граф $(V_1 \cap V_2, E_1 \cap E_2)$.

Дополнением (*дополнительным графом*) к графу $G = (V, E)$ называется граф $\bar{G} = (V, \bar{E})$, где $\bar{E}$ – дополнение множества E до множества всех неупорядоченных пар вершин. Иначе говоря, две различные вершины смежны в графе $\bar{G}$ тогда и только тогда, когда они не смежны в графе G. Например, $\overline{O_n} = K_n$.

Под *суммой* $G_1 + G_2$ двух абстрактных графов понимают объединение графов с непересекающимися множествами вершин. Точнее говоря, имеется в виду следующее. Сначала вершинам графов-слагаемых присваиваются имена (пометки, номера) так, чтобы множества вершин не пересекались, затем полученные графы объединяются и пометки стираются (то есть результат операции – тоже абстрактный граф). Операция сложения ассоциативна, то есть $G_1 + (G_2 + G_3) = (G_1 + G_2) + G_3$ для любых трех графов. Поэтому можно образовывать сумму любого числа графов, не указывая порядка действий с помощью скобок. Если складываются k экземпляров одного и того же графа G, то полученный граф обозначается через kG. Например, $O_n \cong nK_1$.

Соединением графов G_1 и G_2 называется граф $G_1 \circ G_2$, получаемый из их суммы добавлением всех ребер, соединяющих вершины первого слагаемого с вершинами второго. Например, $K_{p,q} = O_p \circ O_q$. На рисунке 1.3 слева изображен граф $C_4 + 2K_2 + 3K_1$, справа – граф $P_3 \circ (P_1 + P_2)$.

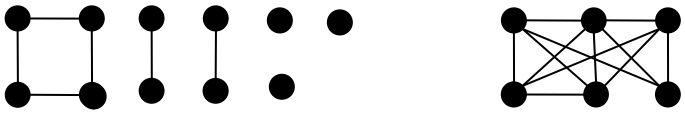


Рис. 1.3. Сумма и соединение графов

Декартово произведение (далее просто «произведение») $G_1 \times G_2$ графов $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ определяется следующим образом. Множеством вершин графа $G_1 \times G_2$ является декартово произведение множеств V_1 и V_2, то есть вершины этого графа – упорядоченные пары (x, y), где $x \in V_1$, $y \in V_2$. Вершины (x_1, y_1) и (x_2, y_2) в графе $G_1 \times G_2$ смежны тогда и только тогда, когда $x_1 = x_2$ и y_1 смежна с y_2 в графе G_2, или $y_1 = y_2$ и x_1 смежна с x_2 в графе G_1. С помощью операции произведения можно выразить некоторые важные графы через простейшие. Например, произведение двух путей дает *прямоугольную решетку* – см. рисунок 1.4.

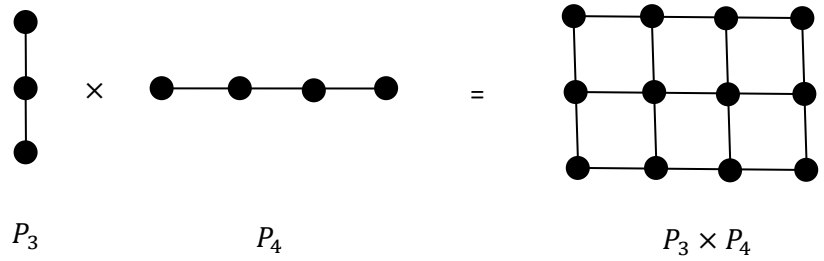


Рис. 1.4. Декартово произведение путей

Другой пример – k-мерный куб Q_k, определяемый следующим образом. Вершинами его являются всевозможные упорядоченные двоичные наборы длины k. Всего, таким образом, в этом графе имеется 2^k вершин. Вершины $x = (x_1, \dots, x_k)$ и $y = (y_1, \dots, y_k)$ смежны в нем тогда и только тогда, когда наборы x и y различаются точно в одной координате. С помощью операции произведения граф Q_k можно определить рекурсивно:

$$Q_1 = K_2, \quad Q_k = Q_{k-1} \times K_2.$$

На рисунке 1.5 показано, как получается Q_3 из Q_2.

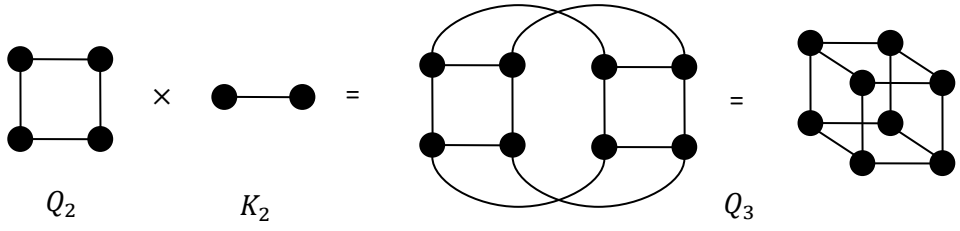


Рис. 1.5. 3-мерный куб как декартово произведение

1.8. Пути, циклы, связность

Путь в графе – это последовательность вершин $x_1, x_2, \dots, x_k$, в которой каждая пара (x_i, x_{i+1}), $i = 1,2, \dots, k-1$, является ребром, причем все эти ребра различны. Путь *соединяет* вершины x_1 и x_k. *Длиной пути* называется число ребер в нем, т.е. $k-1$.

Цикл – путь, у которого $x_1 = x_k$.

Теорема 1.2 (о существовании цикла). *Если в графе степень каждой вершины больше или равна 2, то в нем есть цикл.*

Доказательство. Пусть $P = (x_1, x_2, \dots, x_k)$ – путь наибольшей длины в графе. Так как степени всех вершин не меньше 2, то вершина x_k смежна, кроме вершины x_{k-1}, еще с какой-нибудь вершиной y. Вершина y не может отличаться от всех вершин пути P, иначе образовался бы более длинный путь $x_1, x_2, \dots, x_k, y$. Значит, $y = x_i$ при некотором $i < k-1$. Но тогда $x_i, x_{i+1}, \dots, x_k, x_i$ – цикл. □

Граф называется *связным*, если для любых двух его вершин имеется путь, соединяющий эти вершины. Если граф несвязен, то он состоит из нескольких связных подграфов, между которыми нет ребер, они называются *компонентами связности*.

Теорема 1.3 (о числе ребер в связном графе). *Если граф с n вершинами и m ребрами связен, то $m \geq n-1$.*

Доказательство. Всякий граф с n вершинами можно получить из пустого графа с n вершинами добавлением ребер. Что происходит с компонентами связности графа при добавлении к нему одного нового ребра? Если это ребро соединяет две вершины из одной компоненты связности, то число компонент не изменяется. Если же эти вершины принадлежат разным компонентам, то эти две компоненты объединятся в одну и общее число компонент уменьшится на 1. Таким образом, при добавлении ребра число компонент либо остается прежним, либо уменьшается на 1. Следовательно, для того, чтобы из пустого графа (n компонент) получить связный граф (одна компонента) нужно добавить не менее чем $n-1$ ребро. □

Шарнир (*точка сочленения*) в графе – вершина, при удалении которой увеличивается число компонент связности.

Перешеек – ребро, при удалении которого увеличивается число компонент связности. Легко доказать следующее утверждение.

Теорема 1.4 (о перешейках). *Ребро является перешейком тогда и только тогда, когда через него не проходит ни один цикл.*

В ориентированном графе можно определить два типа путей.

Ориентированный путь – это, как и в неориентированном случае, такая последовательность вершин $x_1, x_2, \dots, x_k$, что для каждого $i = 1,2, \dots, k-1$ пара (упорядоченная) (x_i, x_{i+1}), является ребром и все эти ребра различны.

Неориентированный путь – последовательность вершин и ребер $x_1, e_1, x_2, e_2, \dots, e_{k-1}, x_k$, где $x_1, x_2, \dots, x_k$ – вершины, а $e_1, e_2, \dots, e_{k-1}$ – различные ребра графа, причем $e_i = (x_i, x_{i+1})$ или $e_i = (x_{i+1}, x_i)$ для каждого $i = 1,2, \dots, k-1$.

Орграф называется *связным*, если между любыми двумя вершинами имеется соединяющий их неориентированный путь. Он называется *сильно связным*, если из любой вершины в любую другую имеется ориентированный путь.

1.9. Расстояния и метрические характеристики

Расстоянием между двумя вершинами в графе называется наименьшая длина соединяющего их пути. Расстояние между вершинами x и y обозначается через $d(x, y)$. Если в графе нет пути, соединяющего вершины x и y, то расстояние между ними считается бесконечным или не определенным. Далее (в этом разделе) считаем, что граф связен, тогда функция $d(x, y)$ определена для любой пары вершин. Она обладает свойствами:

1) $d(x, y) \geq 0$, причем $d(x, y) = 0$ тогда и только тогда, когда $x = y$;

2) $d(x, y) = d(y, x)$ для любых вершин x и y;

3) $d(x, y) + d(y, z) \geq d(x, z)$ для любых вершин x, y, z.

Первые два свойства очевидны, третье тоже легко понять: в левой части неравенства записана длина пути из x в y, проходящего через вершину z, а в правой – длина кратчайшего пути между x и y. Это неравенство называется *неравенством треугольника.*

Эксцентриситет вершины – расстояние от нее до самой удаленной вершины:

$$\mathrm{ecc}(x) = \max_{y \in V} d(x, y).$$

Диаметр графа – максимальное расстояние между вершинами, то есть наибольший эксцентриситет:

$$\mathrm{diam}(G) = \max_{x \in V, y \in V} d(x, y) = \max_{x \in V} \mathrm{ecc}(x).$$

Радиус графа – наименьший эксцентриситет:

$$\mathrm{rad}(G) = \min_{x \in V} \mathrm{ecc}(x).$$

Центральная вершина – вершина, эксцентриситет которой равен радиусу графа.

Центр – множество всех центральных вершин. Центр графа G обозначается $C(G)$.

Теорема 1.5. *Для любого связного графа G выполняются неравенства*

$$\mathrm{rad}(G) \leq \mathrm{diam}(G) \leq 2\mathrm{rad}(G).$$

Доказательство. Левое неравенство очевидно, так как радиус – наименьший из эксцентриситетов вершин, а диаметр – наибольший. Докажем правое. Выберем такие вершины a и b, что $d(a,b) = \mathrm{diam}(G)$ и такую вершину c, что $ecc(c) = rad(G)$. Применяя неравенство треугольника и учитывая, что $d(a,c) \leq ecc(c)$ и $d(b,c) \leq ecc(c)$, получаем

$$\mathrm{diam}(G) = d(a,b) \leq d(a,c) + d(c,b) \leq 2\mathrm{ecc}(c) = 2\mathrm{rad}(G).$$

1.10. Графы пересечений

Пусть дано семейство множеств $F = \{S_1, S_2, \dots, S_n\}$. *Графом пересечений* этого семейства называется граф с множеством вершин $\{1,2,\dots,n\}$, в котором вершины i и j смежны тогда и только тогда, когда $S_i \cap S_j \neq \emptyset$. Этот граф обозначается $\Gamma(F)$. Граф $\Gamma(F)$ содержит в компактном виде информацию о том, какие множества из семейства F имеют непустое пересечение. С другой стороны, семейство F можно рассматривать как еще один способ представления графа $\Gamma(F)$. Следующая теорема показывает, что этот способ универсален, то есть любой граф можно представить как граф пересечений некоторого семейства множеств.

Теорема 1.6 (о графах пересечений). *Для любого графа G существует такое семейство множеств F, что $G \cong \Gamma(F)$.*

Доказательство. Пусть $G = (V, E)$, а $V = \{a_1, a_2, \dots, a_n\}$. Для каждой вершины a_i определим множество E_i всех ребер, инцидентных этой вершине. Рассмотрим семейство множеств $F = \{E_1, E_2, \dots, E_n\}$. Две вершины a_i и a_j смежны тогда и только тогда, когда имеется ребро, инцидентное обеим этим вершинам, то есть когда $E_i \cap E_j \neq \emptyset$. Следовательно, $G \cong \Gamma(F)$. □

Разновидностью графов пересечений являются *реберные графы*. Реберный граф графа G – это граф пересечений его ребер, рассматриваемых как множества из двух элементов, он обозначается через $L(G)$. Таким образом, вершины графа $L(G)$ соответствуют ребрам графа G и две вершины в $L(G)$ смежны, если соответствующие ребра в G имеют общую вершину.

Задачи

1.1. Граф задан множеством вершин $V = \{a, b, c, d, e, f\}$ и множеством ребер $E = \{(a, c), (a, f), (b, c), (c, d), (d, f)\}$. Нарисуйте этот граф, постройте для него матрицы смежности и инцидентности, списки смежности.

1.2. Постройте матрицу инцидентности для графа, заданного списками смежности:

$$a: b, d; \quad b: a, c, d, f; \quad c: b, f; \quad d: a, b, f; \quad e: ; \quad f: b, c, d.$$

1.3. В графе 30 вершин и 80 ребер, каждая вершина имеет степень 5 или 6. Сколько в нем вершин степени 5?

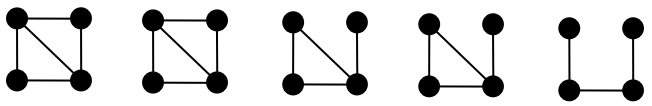
1.4. В графе каждая вершина имеет степень 3, а число ребер заключено между 16 и 20. Сколько вершин в этом графе?

1.5. Найдите все абстрактные графы с 4 вершинами.

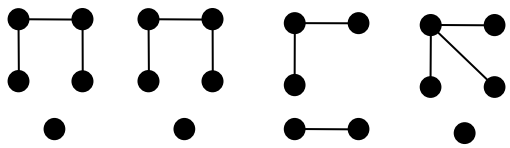
1.6. Найдите все абстрактные графы с набором степеней а) (2,2,2,3,3,4); б) (2,2,2,3,3,3).

1.7. Восстановите граф по его

а) порожденным подграфам, полученным удалением одной вершины:



б) остовным подграфам, полученным удалением одного ребра:



1.8. Граф G имеет множество вершин $\{1,2, \dots, n\}$. Число ребер в подграфе, полученном удалением вершины i, равно m_i, $i = 1,2, \dots, n$. Сколько ребер в графе G?

1.9. Граф имеет n вершин и m ребер. Сколько у него различных
а) остовных подграфов?
б) порожденных подграфов?

1.10. 1) Представьте граф C_6 как объединение трех графов с множествами вершин $\{1,2,3,4\}$, $\{1,2,5,6\}$, $\{3,4,5,6\}$.
2) Верно ли, что любой граф с 6 вершинами можно представить как объединение трех графов с такими множествами вершин?
3) Верно ли, что любой граф с 6 вершинами можно представить как объединение трех графов с множествами вершин $\{1,2,3\}$, $\{3,4,5\}$, $\{5,6,1\}$?

1.11. Какие из следующих высказываний верны?
1) Если H – порожденный подграф графа G, то $\overline{H}$ – порожденный подграф графа $\bar{G}$.
2) Если H – порожденный подграф графа G, то $\bar{G}$ – порожденный подграф графа $\overline{H}$.
3) Если H – остовный подграф графа G, то $\overline{H}$ – остовный подграф графа $\bar{G}$.
4) Если H – остовный подграф графа G, то $\bar{G}$ – остовный подграф графа $\overline{H}$.

1.12. Какие из следующих утверждений верны для любых графов G и H?
1) Граф $G \cap H$ является подграфом графа G.
2) Граф $G \cap H$ – порожденный подграф графа G.
3) Если множества вершин графов G и H совпадают, то $G \cap H$ – остовный подграф графа G.
4) Если H – подграф графа G, то $G \cap H = H$.

1.13. Какие из следующих утверждений верны для любых графов G и H?
1) Граф G является подграфом графа $G \cup H$.
2) Граф G – остовный подграф графа $G \cup H$.
3) Если множества вершин графов G и H не пересекаются, то H – порожденный подграф графа $G \cup H$.
4) Если H – подграф графа G, то $G \cup H = G$.

1.14. Какие из следующих равенств выполняются для любых графов G_1 и G_2?
1) $G_1 \cap (G_1 \cup G_2) = G_1$.
2) $G_1 \cup (G_1 \cap G_2) = G_1$.
3) $\overline{G_1 \cup G_2} = \overline{G_1} \cap \overline{G_2}$.

1.15. Найдите граф G с минимальным числом вершин $n > 1$ такой, что оба графа G и $\bar{G}$ связны.

1.16. Найдите граф G с минимальным числом вершин $n > 1$ такой, что оба графа G и $\bar{G}$ несвязны.

1.17. Сколько вершин и ребер в графе $P_7 \times K_{2,3}$?

1.18. Изоморфны ли графы 1) $P_2 \times C_3$ и $K_2 \circ \overline{2K_2}$; 2) $\overline{K_{1,2}} \times C_3$ и $K_3 + \overline{C_6}$; 3) $\overline{K_4 \times P_2}$ и Q_3?

1.19. Найдите граф с минимальным числом вершин $n > 1$, который не является суммой или соединением меньших графов.

1.20. Сколько в графе K_n имеется а) простых путей длины k; б) простых циклов длины k?

1.21. Сколько в графе K_n имеется подграфов, изоморфных а) графу P_k; б) графу C_k?

1.22. Сколько в графе K_n имеется путей длины а) 2; б) 3; в) 4?

1.23. Граф задан матрицей смежности:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Постройте для него матрицу расстояний между вершинами, найдите эксцентриситеты вершин, диаметр, радиус, центр. Изоморфны ли этот граф и дополнительный к нему?

1.24. Каково расстояние между вершинами (0,0,..., 0) и (1,1,...,1) в графе Q_k? Сколько имеется кратчайших путей между этими вершинами?

1.25. Какой наибольший диаметр может быть у графа с n вершинами? Сколько имеется (абстрактных) графов с таким диаметром?

1.26. Найдите все (с точностью до изоморфизма) графы с 5 вершинами диаметра 3.

1.27. Может ли радиус графа в результате добавления одного нового ребра а) увеличиться; б) уменьшиться; в) остаться прежним?

1.28. Найдите все (с точностью до изоморфизма) графы с 5 вершинами радиуса 1.

1.29. Найдите все (с точностью до изоморфизма) графы с 4 вершинами, имеющие точно одну центральную вершину.

1.30. Найдите диаметр и радиус графа $C_3 \times C_5$.

1.31. Постройте граф пересечений а) граней трехмерного куба; б) ребер графа K_4.

1.32. Сколько ребер в графе $L(K_n)$?

1.33. Найдите количество вершин и ребер в реберном графе $L(G)$ графа G с набором степеней (3,3,3,3,4,4,5,5).

1.34. Приведите пример графа G, для которого $L(G) \cong G$.

1.35. Приведите пример двух неизоморфных графов, у которых реберные графы изоморфны.

1.36. Граф пересечений семейства интервалов на прямой называют графом интервалов. Какие из следующих графов являются графами интервалов: $C_3, P_4, C_4, K_4, C_5, K_{2,3}, K_{1,5}$?

1.37. Граф пересечений семейства дуг окружности называют графом дуг. Какие из графов предыдущей задачи являются графами дуг?

1.38. Докажите, что самодополнительный граф (граф, изоморфный своему дополнению) с n вершинами существует тогда и только тогда, когда $n \equiv 0 \pmod 4$ или $n \equiv 1 \pmod 4$.

1.39. Может ли самодополнительный 100-вершинный граф иметь ровно одну вершину степени 50?

1.39*. Докажите, что если в графе нет порожденного подграфа, изоморфного P_3, то каждая его компонента связности является полным подграфом.

1.40**. Докажите, что если в графе нет порожденного подграфа, изоморфного P_4, то один из графов $G, \bar{G}$ несвязен.

1.41*. Докажите, что если в графе n вершин и n ребер, то в нем есть цикл.

1.42*. Докажите теорему о перешейках.

2. Перечисление графов

2.1. Помеченные графы

Число помеченных обыкновенных графов с множеством вершин V_n обозначим через g_n.

Теорема 2.1 (о числе помеченных графов). $g_n = 2^{\frac{n(n-1)}{2}}$.

Доказательство. Графы с одним и тем же множеством вершин V_n различаются между собой только множествами ребер. Множество ребер такого графа – это какое-нибудь подмножество множества всех неупорядоченных пар различных элементов из V_n, т.е. сочетаний из n по 2. Имеется ровно $\binom{n}{2} = \frac{n(n-1)}{2}$ таких пар. Любое подмножество множества всех пар определяет некоторый граф. Всего имеется $2^{\frac{n(n-1)}{2}}$ подмножеств, это и есть число графов.

2.2. Непомеченные графы

Перечисление непомеченных графов является более трудной задачей. Число абстрактных графов с n вершинами обозначим через $\widetilde{g_n}$. При выводе формулы для этого числа используются некоторые понятия, которые описываются ниже и некоторые факты, которые здесь приводятся без доказательства.

Подстановки. *Подстановкой* называется биекция множества на себя. Подстановка π на множестве V_n записывается в виде таблицы

$$\begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix},$$

при этом $(\pi(1), \pi(2), \dots, \pi(n))$ – перестановка элементов $1,2,\dots,n$. Подстановка на конечном множестве V может быть представлена ориентированным графом с множеством вершин V и ребрами $(x, \pi(x))$ для всех $x \in V$. В этом графе у каждой вершины x степени исхода и захода равны 1. Поэтому граф подстановки состоит из непересекающихся ориентированных циклов (в нем могут быть петли – это циклы длины 1). На рисунке 2.1 показан граф подстановки

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 8 & 3 & 7 & 5 & 9 & 1 & 2 & 6 \end{pmatrix}.$$

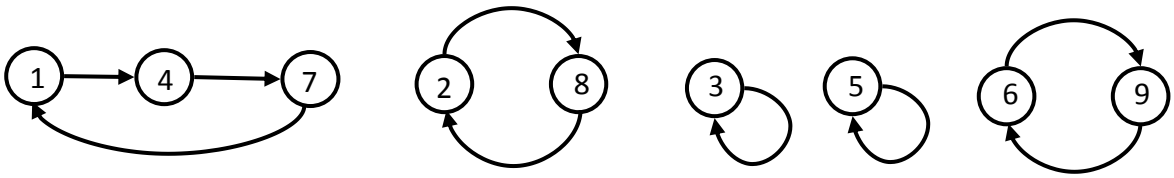


Рис. 2.1. Граф подстановки

Набор чисел $(k_1, k_2, \dots, k_n)$, где k_1 – количество циклов длины 1, k_2 – циклов длины 2, ..., k_n – циклов длины n, называется *цикловой структурой* подстановки. Сумма длин всех циклов равна n, поэтому должно выполняться равенство $k_1 + 2k_2 + \cdots + nk_n = n$. Число подстановок на множестве из n элементов с цикловой структурой $(k_1, k_2, \dots, k_n)$ известно, оно равно

$$w(k_1, k_2, \dots, k_n) = \frac{n!}{k_1!\, k_2! \dots k_n!\, 1^{k_1} 2^{k_2} \dots n^{k_n}}.$$

Автоморфизмы. *Автоморфизм* – подстановка на множестве вершин графа, сохраняющая отношение смежности (две вершины смежны тогда и только тогда, когда смежны их образы). Иначе говоря, автоморфизм – это изоморфизм графа на себя. Тождественная подстановка (каждый элемент переходит в себя) является автоморфизмом любого графа. Произведение $\alpha\beta$ двух автоморфизмов α и β графа G (сначала выполняется подстановка β, потом подстановка α) – тоже автоморфизм этого графа. Для любого автоморфизма α графа G обратное отображение α^{-1} – тоже автоморфизм этого графа. Поэтому множество автоморфизмов данного графа G образует группу подстановок, она обозначается $\mathrm{Aut}\, G$.

Сколько существует различных (помеченных) графов, изоморфных данному графу G ? Иначе говоря: сколько различных графов можно получить, переставляя пометки вершин у графа G? Это зависит от графа: из графа P_4 можно получить 12 разных графов, из C_4 – 3, а для графа K_4 любая перестановка дает тот же самый граф.

Теорема 2.2 (о числе способов пометить граф). *Для любого графа G с n вершинами число различных изоморфных ему графов с тем же множеством вершин равно*

$$\frac{n!}{|\mathrm{Aut}\, G|}.$$

Доказательство. Пусть $G_1 = G, G_2, \dots, G_k$ – все различные графы, изоморфные графу G . Возьмем какую-нибудь подстановку α , преобразующую граф G в граф G_i и пусть β – другая такая подстановка. Рассмотрим подстановку $\gamma = \alpha^{-1}\beta$. Она действует на граф G так: сначала β преобразует граф G в граф G_i, потом α^{-1} преобразует граф G_i обратно в

граф G. Значит, γ – автоморфизм графа G. При этом $\beta = \alpha\gamma$. Таким образом, любая подстановка β, преобразующая граф G в граф G_i, может быть представлена как произведение $\beta = \alpha\gamma$, где $\gamma \in \mathrm{Aut}\, G$. Верно и обратное – любая подстановка вида $\beta = \alpha\gamma$, где $\gamma \in \mathrm{Aut}\, G$ преобразует граф G в граф G_i. Следовательно, число подстановок, преобразующих граф G в граф G_i, равно $|\mathrm{Aut}\, G|$ для любого $i = 1,2,\dots,k$. Всего имеется $n!$ различных подстановок на n вершинах каждая из них преобразует граф G в какой-нибудь из графов G_i, следовательно, $k|\mathrm{Aut}\, G| = n!$. □

Абстрактный граф – это класс эквивалентности графов по отношению изоморфизма. Теорема 2.2 утверждает, что класс эквивалентности, содержащий граф G, состоит из $\frac{n!}{|\mathrm{Aut}\, G|}$ элементов.

Орбиты пар. Пусть π – подстановка на множестве V_n. Относительно этой подстановки множество всех неупорядоченных пар различных элементов из V_n разбивается на *орбиты пар* – орбита состоит из пар, которые переходят друг в друга под действием подстановки π. Например, для подстановки

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix}$$

имеется три орбиты:

(1,2) → (2,3) → (1,3) →(1,2),

(1,4) → (2,5) → (3,4) → (1,5) → (2,4) → (3,5)→(1,4) ,

(4,5)→(4,5) .

Если подстановка π является автоморфизмом графа G, то все пары вершин из одной орбиты одновременно смежны или несмежны в этом графе. Обозначим число орбит пар относительно подстановки π через $c(\pi)$. Любой граф, для которого эта подстановка является автоморфизмом, можно получить следующим образом: каждой орбите приписать одно из значений 0 или 1. Все пары вершин из орбиты объявить смежными, если этой орбите приписана 1, и несмежными, если ей приписан 0. Следовательно, число графов, для которых π является автоморфизмом, равно $2^{c(\pi)}$.

Число $c(\pi)$ зависит только от цикловой структуры подстановки. Оно равно

$$c(\pi) = c(k_1, k_2, \dots, k_n) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} k_i k_j d(i,j) - \frac{1}{2}\sum_{i=1}^{n} k_i \sigma(i),$$

где $d(x,y)$ – наибольший общий делитель чисел x и y, $\sigma(i)$ – остаток от деления числа i на 2.

Число графов. Построим граф Г, имеющий вершины двух типов. Вершины первого типа соответствуют графам с множеством вершин V_n, вершины второго – подстановкам на этом множестве. Вершину-граф соединим ребром с вершиной-подстановкой, если эта подстановка является автоморфизмом этого графа. Подсчитаем двумя способами число ребер в графе Г. С одной стороны, подстановка π является автоморфизмом $2^{c(\pi)}$ графов. Поэтому число ребер в графе Г равно

$$\sum_{\pi} 2^{c(\pi)},$$

суммирование ведется по всем подстановкам на множестве V_n. С другой стороны, из вершины графа Г, соответствующей графу G, выходит $|Aut\ G|$ ребер. Объединим все графы, изоморфные графу G, в один класс, представляющий один абстрактный граф. Из каждой вершины-графа этого класса выходит $|Aut\ G|$ ребер, а всего в классе имеется $\frac{n!}{|\mathrm{Aut}\ G|}$. графов. Следовательно, всего из вершин этого класса выходит $n!$ ребер. Так как имеется $\widetilde{g_n}$ классов, то число ребер в графе Г равно $n!\,\widetilde{g_n}$. Получаем равенство

$$\widetilde{g_n} = \frac{1}{n!}\sum_{\pi} 2^{c(\pi)}.$$

Учитывая формулу для $c(\pi)$, получаем формулу для числа абстрактных графов.

Теорема 2.3 (о числе непомеченных графов).

$$\widetilde{g_n} = \frac{1}{n!} \sum_{\substack{k_1,k_2\ldots,k_n\geq 0,\\ k_1+2k_2+\cdots+nk_n=n}} w(k_1,k_2,\ldots,k_n)2^{c(k_1,k_2,\ldots,k_n)}.$$

2.3. Оценки числа графов

Многие задачи о подсчете числа графов с теми или иными свойствами оказываются очень трудными. И даже когда удается получить точное решение такой задачи, оно может выражаться громоздкой формулой, из которой нелегко понять, например, с какой скоростью растет число графов с данным свойством при увеличении числа вершин. Примером может служить приведенная выше формула для числа абстрактных графов. В таких случаях могут быть полезны оценки числа графов. В качестве примера приведем простую оценку числа абстрактных графов.

Теорема 2.4 (оценка числа непомеченных графов).

$$\widetilde{g_n} \geq \frac{g_n}{n!}.$$

Доказательство. Из любого непомеченного графа с n вершинами можно получить не более чем $n!$ различных помеченных графов, присваивая вершинам в качестве пометок элементы множества V_n. Поэтому $g_n \leq n!\,\widetilde{g_n}$. □

Эта оценка не очень точна при малых n. Например, теорема утверждает, что $\widetilde{g_3} \geq \frac{4}{3}$ и $\widetilde{g_4} \geq \frac{8}{3}$, на самом же деле $\widetilde{g_3} = 4$, $\widetilde{g_4} = 11$. Но при больших n она дает хорошее представление о поведении функции $\widetilde{g_n}$: можно доказать, что отношение левой и правой частей неравенства из теоремы стремится к 1 при $n \to \infty$. Этот факт записывают так:

$$\widetilde{g_n} \sim \frac{g_n}{n!}$$

и говорят, что функции $\widetilde{g_n}$ и $\frac{g_n}{n!}$ *асимптотически равны*.

Говорят, что *почти все* графы обладают некоторым свойством, если отношение числа помеченных графов с n вершинами, имеющих это свойство, к числу всех помеченных графов с n вершинами (т.е. к g_n) стремится к 1 при $n \to \infty$.

Теорема 2.5. *Почти все графы имеют диаметр 2*.

Доказательство. Диаметр каждого графа либо равен 1, либо равен 2, либо больше 2. Диаметр 1 имеет только полный граф. Обозначим через a_n число графов с множеством вершин V_n, имеющих диаметр 2, а через b_n число графов, диаметр которых больше 2. Тогда $g_n = 1 + a_n + b_n$. Следовательно,

$$\frac{a_n}{g_n} = 1 - \frac{1}{g_n} - \frac{b_n}{g_n}.$$

Очевидно, $\frac{1}{g_n} \to 0$. Покажем, что $\frac{b_n}{g_n} \to 0$.

Пусть $x, y \in V_n$ – две вершины. Подсчитаем число графов, у которых расстояние между этими вершинами больше 2. Такой граф можно построить так. Сначала построим какой-нибудь граф на остальных вершинах, т.е. с множеством вершин $V_n - \{x, y\}$. Таких графов имеется ровно $g_{n-2} = 2^{\binom{n-2}{2}}$. Теперь добавим ребра, соединяющие вершины этого графа с вершинами x и y. Так как расстояние между x и y больше 2, то ни одна из остальных вершин не может быть смежной с обеими вершинами x и y. Значит, для каждой вершины из множества $V_n - \{x, y\}$ имеется 3

возможности соединения ее с x и y: можно соединить только с x, или только с y, или ни с одной из них. По правилу произведения для $n-2$ вершин имеется 3^{n-2} возможностей. Следовательно, всего имеется ровно $3^{n-2}2^{\binom{n-2}{2}}$ графов, у которых расстояние между вершинами x и y больше 2.

Число пар различных вершин равно $\binom{n}{2}$. Если диаметр графа больше 2, то хотя бы для одной из этих пар расстояние между вершинами пары больше 2. Следовательно,

$$b_n \le \binom{n}{2} 3^{n-2} 2^{\binom{n-2}{2}}.$$

Теперь оценим отношение

$$\frac{b_n}{g_n} \le \binom{n}{2} \frac{3^{n-2}2^{\frac{(n-2)(n-3)}{2}}}{2^{\frac{n(n-1)}{2}}} = \frac{n(n-1)}{2} \cdot \frac{3^{n-2}}{2^{2n-3}} = \frac{4}{9} n(n-1) \left(\frac{3}{4}\right)^n \to 0$$

при $n \to \infty$. Следовательно, $\frac{a_n}{g_n} \to 1$. □

Если граф несвязен, то его диаметр бесконечен или не определен, в любом случае он не равен 2. Поэтому справедливо следующее утверждение.

Следствие. *Почти все графы связны.*

Задачи

2.1. Сколько имеется неориентированных графов с n вершинами, в которых допускаются петли?

2.2. Найдите число ориентированных графов с n вершинами, в которых
а) возможны петли;
б) петель нет;
в) петель нет и каждая пара различных вершин соединена не более чем одним ребром;
г) возможны петли и каждая пара вершин соединена не более чем одним ребром;
д) петель нет и каждая пара различных вершин соединена точно одним ребром.

2.3. Найдите число графов с n вершинами, в которых допускаются ориентированные и неориентированные ребра (но не петли), причем две вершины могут быть соединены не более чем одним ребром.

2.4. Найдите число неориентированных мультиграфов с n вершинами без петель, в которых для каждой пары вершин имеется не более четырех

соединяющих эти вершины ребер.

2.5. Найдите число графов с n вершинами, в которых а) данные k вершин являются изолированными (имеют степень 0); б) нет изолированных вершин. Верно ли, что почти все графы не имеют изолированных вершин?

2.6. Если к графу с $n-1$ вершиной добавить еще одну вершину и соединить ее ребрами со всеми вершинами нечетной степени, то получится граф с n вершинами, в котором степени всех вершин четны. Сколько имеется графов, у которых степени всех вершин четны? Верно ли, что почти все графы имеют вершины нечетной степени?

2.7. Сколько имеется а) непомеченных, б) помеченных графов с 6 вершинами, у которых степень каждой вершины равна 1?

2.8. Сколько различных помеченных графов можно получить, добавляя одно новое ребро к графу C_n?

2.9. Сколько различных абстрактных графов можно получить, добавляя одно ребро к графу C_n?

2.10. Сколько различных помеченных графов можно получить, добавляя одно ребро к графу P_n?

2.11. Сколько различных абстрактных графов можно получить, добавляя одно ребро к графу а)P_7? б) P_8?

2.12. Сколько различных помеченных графов можно получить, перенумеровывая вершины графа а) $K_{1,q}$? б) P_5? в) C_5?

2.13. Сколько существует помеченных графов, у которых степень каждой вершины равна 2 а) с 5 вершинами; б) с 7 вершинами?

2.14. Сколько различных автоморфизмов имеет граф а) P_n; б) C_n; в) $K_{p,q}$; г) Q_3?

2.15. Найдите по возможности малый граф, не имеющий нетривиальных автоморфизмов.

2.16. Опишите единственный нетривиальный автоморфизм графа P_n с помощью формулы (нумерация вершин начинается с 1).

2.17. Выразите формулами два автоморфизма графа C_n, переводящих вершину i в вершину j (нумерация вершин начинается с 0).

2.18. Пусть $(a_1, a_2, \dots, a_n)$ – двоичный вектор. На множестве вершин графа Q_nрассмотрим преобразование, переводящее вершину

$(x_1, x_2, \dots, x_n)$ в вершину $(x_1 \oplus a_1, x_2 \oplus a_2, \dots, x_n \oplus a_n)$ ($\oplus$ – сумма по модулю 2). Докажите, что оно является автоморфизмом этого графа.

2.19. Сколько имеется орбит пар относительно подстановки на множестве из n элементов, переставляющей два элемента и оставляющей остальные неподвижными?

2.20. Примените формулу для числа абстрактных графов при $n = 4$.

2.21*. Сколько имеется помеченных графов с n вершинами, у которых степень каждой вершины равна 1?

2.22*. Докажите, что почти все графы имеют радиус 2.

2.23**. Докажите, что а) почти все графы имеют хотя бы один треугольник (цикл длины 3); б) почти все графы имеют хотя бы один полный подграф из 4 вершин.

3. Важнейшие классы графов

3.1. Деревья

Деревом называется связный граф, не имеющий циклов. Граф без циклов называют *лесом*. Следующая теорема выражает важнейшее свойство деревьев.

Теорема 3.1 (о путях в дереве). *Для любых двух вершин любого дерева в нем имеется единственный путь, соединяющий эти вершины.*

Доказательство. То, что для любых двух вершин в дереве существует соединяющий их путь, сразу следует из определения – дерево есть связный граф.

Допустим, что в некотором дереве существуют два различных пути, P_1 и P_2, соединяющих вершины a и b. Начальные отрезки этих путей совпадают (оба пути начинаются в одной и той же вершине a). Пусть x – последняя вершина этого общего начала, а следующая на пути P_1 вершина, принадлежащая также и пути P_2, есть y (такая вершина существует, поскольку оба пути заканчиваются в одной вершине b). Тогда объединение отрезков путей P_1 и P_2 между вершинами x и y образует цикл. □

Следствия.

1. *Если из дерева удалить любое ребро, то получится несвязный граф.*
2. *Если к дереву добавить любое новое ребро, то образуется цикл.*

Вершину степени 1в дереве называют *висячей вершиной* или *листом*.

Теорема 3.2 (о листьях). *Для любого дерева с* $n > 1$ *справедливы следующие утверждения:*

1) *для каждой вершины дерева любая наиболее удаленная от нее вершина есть лист;*

2) *в дереве имеется не менее двух листьев.*

Доказательство. Пусть a – какая-нибудь вершина дерева, b – вершина, находящаяся на наибольшем расстоянии от a. Если степень вершины b больше 1, то существует вершина x, смежная с b и не принадлежащая единственному пути между a и b. Но единственный путь между a и x проходит через b, поэтому $d(a,x) = d(a,b) + 1$, а это противоречит тому, что b – вершина, наиболее удаленная от a. Этим доказано первое утверждение теоремы.

Для доказательства второго утверждения возьмем в дереве какую-нибудь вершину a и найдем наиболее удаленную от нее вершину b. По

доказанному b – лист. Пусть c – вершина, наиболее удаленная от b. Тогда c – тоже лист и $c \neq b$. Значит, имеем два листа. □

Отметим, что при любом $n > 1$ существует дерево с n вершинами, имеющее ровно два листа – это граф P_n.

В произвольных графах число ребер может варьироваться от 0 до $\frac{n(n-1)}{2}$. В дереве число ребер жестко связано с числом вершин.

Теорема 3.3 (о числе ребер дерева). *Если в дереве n вершин и m ребер, то $m = n - 1$.*

Доказательство. Проведем индукцию по n. При $n = 1$ утверждение очевидно. Пусть $n > 1$. Удалим из дерева какой-нибудь лист. Вместе с ним будет удалено одно ребро. В результате получится дерево с $n - 1$ вершиной. По предположению индукции у него $n - 2$ ребра. Значит, у исходного дерева было $n - 1$ ребро. □

Следствие 1. *Если в графе, не имеющем циклов, n вершин и m ребер и $m = n - 1$, то этот граф – дерево.*

Доказательство. Пусть $G_1, \dots, G_k$ – компоненты связности графа, причем G_i имеет n_i вершин, $i = 1, \dots, k$. Так как циклов нет, то каждая компонента – дерево, значит, число ребер в G_i равно $m_i = n_i - 1$, а всего ребер в графе $\sum_{i=1}^{k}(n_i - 1) = n - k = n - 1$. Следовательно, $k = 1$ и граф связен. □

Следствие 2. *Если в связном графе n вершин и m ребер и $m = n - 1$, то этот граф – дерево.*

Доказательство. Если в связном графе есть цикл, то можно удалить любое ребро этого цикла, полученный граф тоже будет связным. Можно продолжать удалять цикловые ребра до тех пор, пока не останется связный граф без циклов, т.е. дерево. Но число ребер в этом дереве будет меньше, чем $n - 1$, а это противоречит теореме 3.3. □

У графа может быть единственная центральная вершина (пример: граф P_5), но есть графы, у которых все вершины – центральные (полный граф, например). Возможны и разные промежуточные варианты. Для деревьев есть только две возможности.

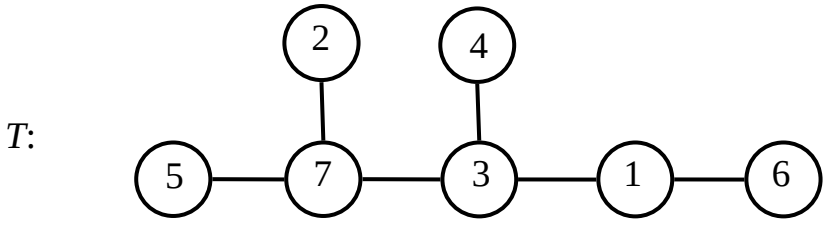
Теорема 3.4 (о центре дерева). *Центр любого дерева состоит из одной вершины или из двух смежных вершин.*

Доказательство. Проведем индукцию по числу вершин n. При $n = 1$ и $n = 2$ утверждение верно. Пусть T– дерево с n вершинами $n > 2$. Удалив из дерева T все листья одновременно, получим дерево T'. Так как для

каждой вершины дерева наиболее удаленная от нее вершина является листом, то при удалении всех листьев эксцентриситет каждой из оставшихся вершин уменьшится на 1. Значит, центральными вершинами нового дерева будут те же, что были центральными вершинами исходного, т.е. $C(T') = C(T)$. Но в вершин в дереве T' меньше и можно применить предположение индукции, согласно которому $C(T')$ состоит из одной вершины или из двух смежных. □

Код Прюфера. Число деревьев. Наиболее компактным способом представления помеченных деревьев является *код Прюфера*. Он строится следующим образом.

Пусть $T -$ дерево с множеством вершин V_n. Найдем в T наименьший лист a_1. Запомним вершину b_1, смежную с a_1 и удалим вершину a_1 из T. В полученном дереве опять найдем наименьший лист a_2, запомним смежную с ним вершину b_2, удалим a_2 и т.д. Эти действия повторяются до тех пор, пока в дереве не останутся две вершины. Последовательность чисел $p(T) = (b_1, b_2, \dots, b_{n-2})$ и есть код Прюфера дерева T. На рисунке 3.1 показано дерево и его код Прюфера.



$p(T)$ = (7, 3, 7, 1, 3)

Рис. 3.1. Дерево и его код Прюфера

По коду Прюфера можно восстановить дерево. Ребрами дерева T являются пары $(a_1, b_1), (a_2, b_2), \dots, (a_{n-2}, b_{n-2})$ и еще одна пара вершин, оставшихся в дереве после окончания процесса кодирования, обозначим эту пару (a_{n-1}, b_{n-1}). Числа $b_1, \dots, b_{n-2}$ известны из кода, остается найти числа $a_1, \dots, a_{n-2}, a_{n-1}, b_{n-1}$.

Составим таблицу, в которой для каждого $x = 1,2, \dots, n$ записано число вхождений x в код $p(T)$. Это число обозначим $k(x)$. Например, для кода $p(T) = (3,6,1,1,3,1)$ получаем таблицу:

X	1	2	3	4	5	6	7	8
$k(x)$	3	0	2	0	0	1	0	0

Для каждой вершины x дерева T выполняется равенство $\deg(x) = k(x) + 1$. В самом деле, вершина x в процессе построения кода либо будет удалена из дерева, либо останется в числе двух последних вершин. В любом случае в какой-то момент она превратится в лист. Это значит, что к

этому моменту из дерева будут удалены $\deg(x) - 1$ из смежных с x вершин. При удалении каждой из этих вершин вершина x заносится в код. После того, как вершина x станет листом, независимо от того, будет ли она впоследствии удалена из дерева или останется в числе двух последних вершин, она более ни одного раза не будет добавлена к коду. Значит, она будет занесена в код ровно $\deg(x) - 1$ раз.

Таким образом, листья – это вершины, для которых $k(x) = 0$. Теперь легко находится наименьший лист a_1 (в примере $a_1 = 2$). Это дает нам ребро (a_1, b_1). Вычеркнем из таблицы столбец, соответствующий a_1, а значение $k(b_1)$ уменьшим на 1. По полученной таблице находим следующий наименьший лист a_2 и т.д. В результате будут найдены ребра $(a_1, b_1), (a_2, b_2), \dots, (a_{n-2}, b_{n-2})$, после чего в таблице останется два столбца. Их номера дадут последнее ребро (a_{n-1}, b_{n-1}). Следующая таблица иллюстрирует процесс декодирования для кода $p(T) = (3,6,1,1,3,1)$. Строчки таблицы представляют значения функции $k(x)$ после очередного преобразования, а последний столбец – восстанавливаемые ребра дерева.

1	2	3	4	5	6	7	8	
3	0	2	0	0	1	0	0	(2,3)
3		1	0	0	1	0	0	(4,6)
3		1		0	0	0	0	(5,1)
2		1			0	0	0	(6,1)
1		1				0	0	(7,3)
1		0					0	(3,1)
0							0	(1,8)

Описанная процедура позволяет по любому набору $(b_1, b_2, \dots, b_{n-2})$ элементов множества V_n найти $n - 1$ пар $(a_1, b_1), (a_2, b_2), \dots, (a_{n-1}, b_{n-1})$. Это множество пар можно рассматривать как множество ребер некоторого графа. Но всегда ли этот граф будет деревом? Ответ утвердительный. Заметим, что в момент, когда по таблице ищется a_i, таблица представляет число вхождений оставшихся элементов в набор $(b_i, \dots, b_{n-2})$. Но в этот момент $k(a_i) = 0$. Значит, a_i не совпадает ни с одним из $b_i, \dots, b_{n-2}$. Ясно также, что a_i отличается от $a_{i+1}, \dots, a_{n-1}$ (вообще все $a_1, a_2, \dots, a_{n-1}$ различны) и от b_{n-1}. Поэтому, если обозначить через G_i граф с множеством ребер $\{(a_i, b_i), (a_{i+1}, b_{i+1}), \dots, (a_{n-1}, b_{n-1})\}$, то вершина a_i в этом графе имеет степень 1. Через вершину степени 1 не проходит никакой цикл. Значит, если в графе G_{i+1} нет циклов, то их нет и в графе G_i.

Граф G_{n-1} не содержит циклов (у него только одно ребро), следовательно, и в графе G_1 циклов нет. Так как в нем $n-1$ ребро, то этот граф – дерево.

Таким образом, функция $p(T)$ устанавливает взаимно однозначное соответствие между множеством всех деревьев с множеством вершин V_n и множеством упорядоченных наборов длины $n-2$, составленных из элементов множества V_n. Отсюда следует формула для числа помеченных деревьев. Обозначим через t_n число деревьев с множеством вершин V_n.

Теорема 3.5 (формула Кэли). $t_n = n^{n-2}$.

Корневые деревья. *Корневым деревом* называют дерево с выделенной вершиной – *корнем*.

Высота корневого дерева – это расстояние от корня до самого удаленного листа.

Если в корневом дереве T путь, соединяющий вершину x с корнем, проходит через вершину y, то говорят, что y – *предок* x, а x – *потомок* y. В частности, каждая вершина является предком и потомком самой себя. Предок (потомок) вершины, отличный от нее самой, называется *собственным* предком (потомком). Множество всех предков вершины порождает путь из корня в эту вершину. Множество всех потомков вершины x порождает дерево с корнем x, оно называется *ветвью* дерева T в вершине x. Если предок и потомок соединены ребром, то они называются соответственно *отцом* и *сыном*. Компактный и полезный во многих случаях способ задать корневое дерево состоит в указании для каждой вершины ее отца (отцом корня иногда считают саму эту вершину).

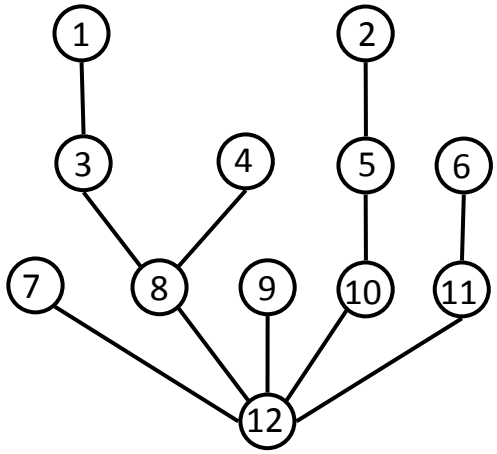
Изоморфизм деревьев. Выяснить, изоморфны ли два графа, оказывается довольно сложной задачей в общем случае. Для деревьев же известно достаточно эффективное решение.

Рассмотрим сначала корневые деревья. Изоморфизм корневых деревьев определяется так же, как изоморфизм графов вообще, но с дополнительным требованием: корень при изоморфизме должен переходить в корень. Проверка изоморфизма корневых деревьев выполняется достаточно легко с помощью специального кодирования. *Канонический код* $c(T)$ дерева T можно построить следующим образом.

Каждой вершине x дерева T приписывается код вершины – слово $c(x)$ в алфавите $\{0,1\}$. Он строится по следующим правилам. Если x – лист (но не корень), то $c(x) = \lambda$ где λ – пустое слово. Код вершины x, не являющейся листом, определяется после того, как построены коды всех ее сыновей. Пусть $y_1, y_2, \dots, y_k$ – эти сыновья, перечисленные в порядке неубывания их кодов: $c(y_1) \le c(y_2) \le \cdots \le c(y_k)$, здесь знак $\le$ обозначает лексикографический порядок. Тогда полагаем

$$c(x) = 0c(y_1)10c(y_2)1 \dots 0c(y_k)1.$$

Код корня объявляется каноническим кодом дерева. На рисунке 3.1 показаны дерево с корнем в вершине 12 и таблица кодов его вершин.



1	λ
2	λ
3	01
4	λ
5	01
6	λ
7	λ
8	010011
9	λ
10	0011
11	01
12	0101000111001100100111

Рис. 3.1. Канонический код дерева

Канонический код любого дерева обладает свойствами:

1) число нулей в нем равно числу единиц;

2) в каждом его префиксе (начальном отрезке) число единиц не превосходит числа нулей.

Первое следует из того, что нули и единицы к кодам сыновей добавляются парами, а второе – из того, что слева всегда добавляется ноль, а справа единица.

Используя эти свойства, можно легко восстановить дерево по его коду. Код дерева T имеет вид: $c(T) = 0c(T_1)10c(T_2)1 \dots 0c(T_k)1$, где $T_1, T_2, \dots, T_k$ – ветви в сыновьях корня. Так как в слове $c(T_1)$ число нулей равно числу единиц, а в каждом его префиксе число нулей не меньше числа единиц, то наименьший префикс слова $c(T)$, в котором число нулей равно числу единиц, – это $0c(T_1)1$. Таким образом, по коду дерева легко находится код первой ветви. Следующее (если двигаться вдоль кода $c(T)$ слева направо) совпадение числа нулей и числа единиц произойдет, когда будет прочитан код второй ветви и т.д. Если код какой-то ветви оказывается пустым словом, то эта ветвь состоит из единственной вершины. К кодам ветвей процедура применяется рекурсивно. Таким образом, по коду $c(T)$ корневое дерево восстанавливается однозначно (с точностью до обозначения вершин, т.е. до изоморфизма).

Теорема 3.6. *Корневые деревья T' и T'' изоморфны тогда и только тогда, когда $c(T') = c(T'')$.*

Доказательство. Так как дерево по коду восстанавливается однозначно с точностью до изоморфизма, то из равенства кодов двух деревьев следует, что эти деревья изоморфны. Обратно, пусть T' и T'' – изоморфные корневые деревья. Докажем индукцией по числу вершин, что их коды совпадают. Пусть $T_1', T_2', \dots, T_k'$ и $T_1'', T_2'', \dots, T_k''$ – ветви в сыновьях корней деревьев T' и T'' соответственно. Тогда существует такая перестановка $(p_1, p_2, \dots, p_k)$ номеров $1,2, \dots, k$, что $T_i' \cong T_{p_i}''$ для каждого $i = 1,2, \dots, k$. По предположению индукции $c(T_i') = c(T_{p_i}'')$. Значит, наборы слов $(c(T_1'), \dots, c(T_k'))$ и $(c(T_1''), \dots, c(T_k''))$ различаются только порядком слов. После того, как каждый из них будет лексикографически упорядочен, эти наборы станут одинаковыми. Поэтому коды $c(T')$ и $c(T'')$ совпадут. □

Описанное кодирование можно применить и для распознавания изоморфизма обычных (не корневых) деревьев. Допустим, нужно сравнить деревья T' и T''. Сначала находим центры обоих деревьев. Ясно, что при изоморфизме центр должен перейти в центр. Если центр каждого из деревьев состоит из одной вершины, выберем эти вершины в качестве корней, затем построим канонические коды корневых деревьев и сравним их. Если центр одного дерева – одна вершина, а центр другого – две, то эти деревья не изоморфны. Допустим, каждый из центров состоит из двух вершин. Если удалить ребро, соединяющее две центральные вершины, то дерево разобьется на два корневых дерева (с корнями в бывших центральных вершинах). Построив коды этих корневых деревьев, получаем для дерева T' пару слов (α', β'), а для дерева T'' – пару слов (α'', β''). Деревья изоморфны тогда и только тогда, когда эти неупорядоченные пары совпадают.

Каркас графа. *Каркас* связного графа – это остовный подграф, являющийся деревом. Применяются также термины *остов*, *остовное дерево*. В общем случае (граф может быть несвязным) каркас определяется как лес, в котором каждая компонента связности является каркасом компоненты связности графа. У любого графа есть хотя бы один каркас. Его можно найти, удаляя цикловые ребра (ребра, принадлежащие циклам) до тех пор, пока все циклы не будут разрушены.

Если в графе есть циклы, то у него больше одного каркаса. Определить точное число каркасов связного графа позволяет так называемая матричная теорема Кирхгофа. Приведем ее без доказательства. Для графа G определим матрицу $k(G)$ – квадратную матрицу порядка n с элементами

$$k_{ij} = \begin{cases} -1, & \text{если вершины } i \text{ и } j \text{ смежны,} \\ 0, & \text{если вершины } i \text{ и } j \text{ не смежны и } i \neq j, \\ \deg(i), & \text{если } i = j. \end{cases}$$

Заметим, что матрица $k(G)$ – вырожденная, так как сумма элементов каждой строки равна 0.

Теорема 3.7 (теорема Кирхгофа). *Если G – связный граф с не менее чем двумя вершинами, то алгебраическое дополнение любого элемента матрицы $k(G)$ равно числу каркасов графа G.*

3.2. Двудольные графы

Граф называется *двудольным*, если множество его вершин можно так разбить на два подмножества (*доли*), чтобы каждое ребро соединяло вершины из разных подмножеств. Иначе говоря, каждое из подмножеств порождает пустой подграф.

Теорема 3.8 (теорема Кёнига, критерий двудольности). *Граф является двудольным тогда и только тогда, когда в нем нет циклов нечетной длины.*

Доказательство. Предположим, G – двудольный граф, $V = V_1 \cup V_2$ – разбиение его множества вершин на две доли и $C = (x_1, x_2, \dots, x_k, x_1)$ – цикл длины k в этом графе. Допустим, вершина x_1 принадлежит доле V_1. Тогда смежная с ней вершина x_2 принадлежит доле V_2, смежная с x_2 вершина x_3 – доле V_1 и т.д. Таким образом, вершина x_i принадлежит доле V_1 при нечетном i и доле V_2 – при четном. Так как вершина x_k смежна с вершиной x_1, то она находится в доле V_2, т.е. k – четное число. Значит, любой цикл в любом двудольном графе имеет четную длину, т.е. условие теоремы является необходимым. Докажем его достаточность.

Очевидно, граф двудольный тогда и только тогда, когда все его компоненты связности – двудольные графы. Поэтому достаточно рассматривать только связные графы. Пусть G – связный граф, не имеющий циклов нечетной длины. Выберем в нем произвольную вершину a и разобьем все вершины графа на два множества: V_1 – те, у которых расстояние от вершины a нечетно, V_2 – те, у которых оно четно. Покажем, что в графе нет ребер, соединяющих вершины из одного множества. Допустим, (b, c) – такое ребро. Рассмотрим два случая.

1) Вершины b и c находятся на разных расстояниях от вершины a. Пусть, для определенности, $d(a, b) < d(a, c)$. Рассмотрим кратчайший путь между вершинами a и b. Поскольку b и c смежны, то добавив к этому пути одно ребро, получим путь, соединяющий вершины a и c. Поэтому $d(a, c) = d(a, b) + 1$. Но это невозможно, так как вершины b и c принадлежат одному из множеств V_1, V_2, следовательно, $d(a, b)$ и $d(a, c)$ имеют одинаковую четность.

2) $d(a, b) = d(a, c)$. Пусть P_b и P_c – кратчайшие пути от вершины a до вершин b и c. Эти пути начинаются в одной вершине, а заканчиваются в

разных. Пусть x – последняя вершина пути P_b, которая принадлежит также пути P_c и пусть $d(a,x) = l$. Рассмотрим отрезки путей P_b и P_c от вершины x до вершин b и c. Каждый из них имеет длину $k - l$. Но тогда соединение этих отрезков и ребра (b,c) образует цикл длины $2(k-l)+1$, а это противоречит условию, что в графе G нет циклов нечетной длины. □

3.3. Планарные графы

Плоский граф – это граф, вершинами которого являются точки плоскости, а ребра представлены линиями, соединяющими смежные вершины, при этом никакие два ребра не должны иметь общих точек, кроме инцидентной им обоим вершины. Граф называется *планарным*, если он изоморфен плоскому графу. Этот плоский граф называют также *плоской укладкой* планарного графа.

Если плоскость разрезать по ребрам плоского графа, она распадется на связные части, которые называют *гранями*. Всегда имеется одна неограниченная *внешняя* грань, все остальные грани называются *внутренними*.

Теорема 3.9 (формула Эйлера). *Количество граней в любой плоской укладке планарного графа, имеющего* n *вершин,* m *ребер и* k *компонент связности, равно* $m - n + k + 1$.

Доказательство. Докажем сначала утверждение теоремы при $k = 1$. Рассмотрим связный плоский граф G. Если в нем нет циклов, то имеется единственная грань, а $m = n - 1$, и формула верна. Если же есть хотя бы один цикл, то возьмем какое-нибудь ребро e, принадлежащее простому циклу C. Это ребро принадлежит границе двух граней, одна из которых целиком лежит внутри цикла C, другая снаружи. Если удалить ребро e из графа, эти две грани сольются в одну. Граф G_1, полученный из графа G удалением ребра e, очевидно, будет плоским и связным, в нем на одно ребро и на одну грань меньше, чем в G, а число вершин осталось прежним. Если в G_1 еще есть циклы, то, удалив еще одно цикловое ребро, получим граф G_2. Будем продолжать удаление цикловых ребер до тех пор, пока не получится связный плоский граф без циклов, т.е. дерево. У него $n - 1$ ребро и единственная грань. Значит, всего было удалено $m - n + 1$ ребро, а так как при удалении каждого ребра число граней уменьшалось на единицу, то в исходном графе было $m - n + 2$ грани. Таким образом, формула верна для любого связного плоского графа. Если граф несвязен, то в компоненте связности, имеющей n_i вершин и m_i ребер, как доказано выше, имеется $m_i - n_i + 1$ внутренняя грань. Суммируя по всем компонентам и прибавляя 1 для учета внешней грани, убеждаемся в справедливости формулы в общем случае. □

Следствие 1. *Если в планарном графе* n *вершин,* $n \geq 3$, *и* m *ребер, то* $m \leq 3(n-2)$.

Доказательство. Если в графе нет циклов, то $m = n - k$ (см. задачу 3.1) и неравенство выполняется при $n \geq 3$. Рассмотрим плоский граф G с r гранями, в котором имеются циклы. Занумеруем грани числами от 1 до r и обозначим через a_i количество ребер, принадлежащих границе грани с номером i. Так как граница каждой грани содержит цикл, то $a_i \geq 3$ для каждого i, следовательно, $\sum_{i=1}^{r} a_i \geq 3r$. С другой стороны, каждое ребро принадлежит границе не более чем двух граней, поэтому $\sum_{i=1}^{r} a_i \leq 2m$. Из этих двух неравенств следует, что $3r \leq 2m$. Применяя формулу Эйлера, получаем $m \leq 3n - 3k - 3 \leq 3n - 6$. □

Следствие 1 дает необходимое условие планарности, которое в некоторых случаях позволяет установить, что граф не является планарным. Рассмотрим, например, полный граф K_5. У него $n = 5$, , $m = 10$ и мы видим, что неравенство из следствия 1 не выполняется. Значит, этот граф непланарен. В то же время существуют графы, не являющиеся планарными, для которых неравенство следствия 1 выполняется. Пример – полный двудольный граф $K_{3,3}$. У него 6 вершин и 9 ребер. Неравенство выполняется, но мы сейчас установим, что он непланарен. Заметим, что в этом графе нет циклов длины 3 (так как он двудольный, в нем вообще нет циклов нечетной длины). Поэтому граница каждой грани содержит не менее четырех ребер. Повторяя рассуждения из доказательства следствия 1, но используя неравенство $a_i \geq 4$ вместо $a_i \geq 3$, получаем следующий результат.

Следствие 2. *Если в планарном графе* n *вершин,* $n \geq 3$, m *ребер и нет циклов длины* 3, *то* $m \leq 2(n-2)$.

Для графа $K_{3,3}$ неравенство следствия 2 не выполняется, и это доказывает, что он непланарен.

Критерии планарности. Известно несколько критериев планарности, сформулируем без доказательства два из них.

Операция *подразбиения* ребра (a, b) действует следующим образом. Из графа удаляется это ребро, к нему добавляется новая вершина c и два новых ребра (a, c) и (b, c). Граф H называется *подразбиением* графа G, если первый можно получить из второго последовательностью подразбиений ребер.

Теорема 3.10 (теорема Понтрягина–Куратовского). *Граф планарен тогда и только тогда, когда у него нет подграфа, являющегося подразбиением графа* K_5 *или графа* $K_{3,3}$.

Операция *стягивания* ребра $(a.b)$ определяется следующим образом. Вершины a и b удаляются из графа, к нему добавляется новая вершина c и она соединяется ребром с каждой вершиной, с которой была смежна хотя бы одна из вершин a и b. Граф G называется *стягиваемым* к графу H, если H можно получить из G последовательностью операций стягивания ребер.

Теорема 3.11 (теорема Вагнера). *Граф планарен тогда и только тогда, когда у него нет подграфа, стягиваемого к графу* K_5 *или графу* $K_{3,3}$.

Алгоритм проверки планарности. Рассмотрим один из алгоритмов проверки планарности. Предполагается, что граф связен и в нем нет шарниров. Алгоритм строит плоскую укладку графа или определяет, что граф не планарный.

Сначала находится и укладывается какой-нибудь простой цикл, при каждой следующей итерации к уложенному подграфу добавляются новые вершины и ребра.

Пусть подграф F есть уложенная к настоящему моменту часть графа, а H – подграф, порожденный не уложенными еще вершинами. *Сегментом* назовем компоненту связности графа H с добавленными к ней *контактными вершинами* – вершинами графа F, смежными с вершинами этой компоненты, и ребрами, соединяющими контактные вершины с вершинами компоненты. Сегментом считается также ребро, соединяющее две вершины подграфа F, но само этому подграфу не принадлежащее, вместе с инцидентными ему вершинами.

В примере на рисунке 3.2 выделены вершины и ребра подграфа F, $\Gamma_1 - \Gamma_4$ – грани этого подграфа. Справа показаны три сегмента относительно этого подграфа.

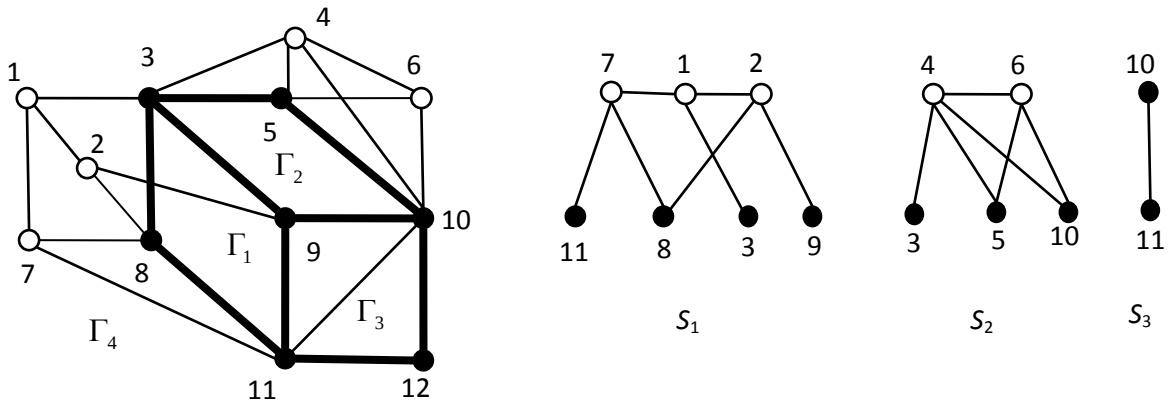


Рис. 3.2. Граф с уложенным подграфом и сегменты

Грань называется *допустимой* для некоторого сегмента, если все контактные вершины этого сегмента принадлежат этой грани. Если

обозначить через $\Gamma(S)$ множество допустимых граней для сегмента S, то в примере имеем $\Gamma(S_1)=\{\Gamma_1\}$, $\Gamma(S_2)=\{\Gamma_2,\Gamma_4\}$, $\Gamma(S_3)=\{\Gamma_3,\Gamma_4\}$.

Алгоритм проверки планарности и построения плоской укладки

1. Найти в графе какой-нибудь простой цикл и уложить его.
2. Пока есть не уложенные ребра, повторять:
   - 2.1. найти грани уложенного подграфа, сегменты и допустимые грани для каждого сегмента;
   - 2.2. если для некоторого сегмента нет допустимых граней, то граф не планарен, стоп;
   - 2.3. если есть сегменты, имеющие только одну допустимую грань, то пусть S – такой сегмент, иначе выбрать любой сегмент S;
   - 2.4. в сегменте S найти путь P, соединяющий две контактные вершины этого сегмента и не содержащий других контактных вершин;
   - 2.5. уложить путь P в одну из допустимых граней для сегмента S.

В применении к рассмотренному примеру в качестве S должен быть выбран сегмент S_1, а в качестве пути P можно взять, например, путь (3, 1, 7, 11). Он должен быть уложен в грань Γ_1, после чего она разобьется на две новых грани. При следующем повторении основного цикла обнаруживается, что имеется сегмент, состоящий из вершины 2 и контактных вершин 1, 8, 9, для которого нет допустимых граней. Значит, этот граф не планарный.

Если работа алгоритма заканчивается построением укладки всего графа, то, очевидно, этот граф планарный. Для обоснования алгоритма необходимо еще доказать обратное утверждение – если граф планарен, то алгоритм построит его плоскую укладку. Доказательство можно найти в книге [5].

Задачи

3.1. Сколько ребер в лесе с n вершинами и k компонентами связности?

3.2. Сколько ребер в связном графе с n вершинами, если в нем имеется единственный цикл?

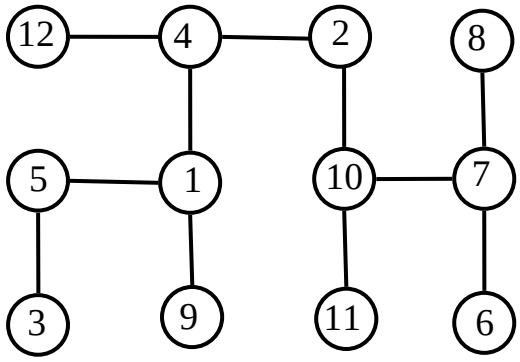
3.3. Перечислите все абстрактные деревья с 6 (7) вершинами.

3.4. В дереве имеется 40 вершин степени 4, все остальные вершины – листья. Сколько листьев в этом дереве?

3.5. В дереве имеется ровно три листа a, b, c, причем $d(a,b)=37$, $d(a,c)=44$, $d(b,c)=21$. Сколько всего вершин в этом дереве?

3.6. Дерево имеет две центральные вершины, а его радиус равен 6. Чему равен диаметр этого дерева?

3.7. Постройте код Прюфера для дерева, изображенного на рисунке.



3.8. Восстановите дерево по коду Прюфера $p(T) = (4, 1, 6, 2, 2, 2, 7, 6)$.

3.9. Сколько имеется помеченных корневых деревьев с n вершинами?

3.10. Сколько различных (помеченных) каркасов у графа K_5?

3.11. Сколько попарно неизоморфных каркасов у графа K_5?

3.12. С помощью теоремы Кирхгофа найдите число каркасов у графа $K_{2,3}$.

3.13. Найдите два неизоморфных дерева с одинаковыми наборами степеней вершин.

3.14. Найдите дерево с наименьшим числом вершин, не имеющее нетривиальных автоморфизмов.

3.15. Постройте канонический код дерева из задачи 3.8, взяв в качестве корня вершину 2.

3.16. Восстановите дерево по каноническому коду $c(T) = 000111001001011100101010 11$.

3.17. Какие из следующих графов являются двудольными: 1) $\overline{C_5}$; 2) $\overline{2C_4}$; 3) $\overline{K_5 + K_6}$; 4) $\overline{2K_2 \circ 2K_2}$?

3.18. Сколько различных абстрактных двудольных графов можно получить, добавляя одно ребро к графу а) P_7; б) P_8; в) C_{12}?

3.19. Какое наименьшее число ребер нужно удалить из графа K_8, чтобы получился двудольный граф?

3.20. Двудольный граф имеет k компонент связности. Каким числом способов его можно разбить на две доли?

3.21. При каких значениях k граф Q_k является двудольным?

3.22. Найдите все двудольные графы с 4 вершинами, у которых дополнительный граф также является двудольным. Докажите, что не существует таких графов с числом вершин больше 4.

3.23. Каково наибольшее число ребер в двудольном графе с n вершинами?

3.24. Какие из следующих графов планарны: 1) $\overline{C_5 + K_1}$; 2) $\overline{K_3 + K_4}$; 3) $\overline{3K_2}$; 4) $\overline{K_1 \circ \overline{K_5}}$?

3.25. Какое наибольшее число граней может быть у плоского графа с 5 вершинами?

3.26. Какое наименьшее число ребер нужно удалить из графа K_6, чтобы получился планарный граф?

3.27. Сколько существует абстрактных непланарных графов с 6 вершинами, имеющих ровно одну вершину степени 3?

3.28. Какое наименьшее количество новых ребер нужно добавить к графу а) C_6; б) C_{100}, чтобы получился непланарный граф?

3.29. Какое наибольшее количество новых ребер можно добавить к графу а) C_6; б) C_{100} так, чтобы граф остался планарным?

3.30. Найдите пример графа G с 8 вершинами такого, что оба графа G и $\bar{G}$ планарные. Докажите, что не существует такого графа с 11 вершинами.

3.31. При каких значениях k граф Q_k является планарным?

3.32. Примените алгоритм распознавания планарности а) к графу из задачи 1.22; б) к графу $\overline{P_6}$; в) к графу $\overline{C_7}$.

3.33. Плоский граф называется *плоской триангуляцией*, если граница каждой грани состоит из трех ребер. Докажите, что каждый плоский граф является подграфом плоской триангуляции. Сколько ребер у плоской триангуляции с n вершинами?

3.34**. Граф называется *расщепляемым*, если множество его вершин можно разбить на два подмножества, одно из которых порождает пустой, другое – полный подграф. Докажите, что граф расщепляемый тогда и только тогда, когда в нем нет порожденного подграфа, изоморфного какому-нибудь из графов $2K_2$, C_4, , C_5.

3.35*. Докажите, что почти все графы не двудольные.

3.36**. Докажите, что почти все графы не планарные.

3.37**. Покажите, что алгоритм построения кода Прюфера можно

реализовать так, что время его работы будет $O(n)$.

3.38**. Покажите, что алгоритм восстановления дерева по коду Прюфера можно реализовать так, что время его работы будет $O(n)$.

4. Методы обхода графа

Решение многих задач на графах основывается на полном обходе графа. Такой обход можно выполнить многими способами, но наибольшее распространение получили две стратегии – поиск в ширину и поиск в глубину. Оба эти метода можно рассматривать как реализации общего плана обхода, состоящего в следующем.

Обход начинается в заранее выбранной стартовой вершине и состоит в систематическом исследовании ребер и посещении вершин. Какие именно действия выполняются при этом, зависит от конкретной задачи, для решения которой и выполняется обход. Но в любом случае тот факт, что данная вершина посещена, запоминается. Вершину, которая еще не посещена, будем называть *новой*. В результате посещения вершина становится *открытой* и остается такой, пока не будут исследованы все инцидентные ей ребра. После этого она превращается в *закрытую*.

Очередной шаг обхода начинается с выбора какой-либо вершины x из множества открытых, она становится *активной*. Если среди инцидентных ей ребер имеются неисследованные, то выбирается такое ребро (x, y). Если вершина y новая, то она посещается, ребро (x, y) при этом классифицируется как *прямое*. Если же y не новая, то ребро (x, y) считается *обратным* (ведущим в уже посещенную вершину). Если все ребра, инцидентные активной вершине, уже исследованы, она становится закрытой. Эти действия повторяются до тех пор, пока множество открытых вершин не станет пустым. Если при этом еще остались новые вершины, то выбирается и посещается одна из таких вершин и процесс повторяется.

По окончании обхода прямые ребра образуют каркас графа. В основе большинства применений поиска в ширину и поиска в глубину лежат свойства этого каркаса.

Основное различие между поиском в ширину и поиском в глубину состоит в том, как выбирается активная вершина.

4.1. Поиск в ширину

При поиске в ширину в качестве активной вершины выбирается та из открытых, которая была посещена раньше других. Для реализации такого правила выбора удобно использовать очередь для хранения множества открытых вершин.

В приводимом описании процедуры поиска в ширину (BFS – Breadth First Search – английское название поиска в ширину) a – стартовая вершина, Q – очередь открытых вершин. Напомним, что $N(x)$ обозначает множество вершин, смежных с вершиной x.

Procedure BFS(a)

1 посетить a;
2 **while** $Q \neq \emptyset$ **do**
3 взять x из Q;
4 **for** $y \in N(x)$ **do**
5 **if** y новая **then** посетить y

Здесь «взять x из Q» означает, что переменной x присваивается значение, равное первому элементу в очереди Q, а сам этот элемент из очереди удаляется. Процедура посещения, кроме действий, связанных с решением конкретной задачи, должна включать две обязательных операции: вершину нужно пометить как посещенную (т.е. не новую) и поместить в очередь Q.

Процедура BFS обеспечивает посещение всех вершин из компоненты связности, содержащей вершину a. Для полного обхода графа с множеством вершин V нужно добавить еще внешний цикл:

for $x \in V$ **do if** x новая **then** BFS(x).

Оценим время работы алгоритма. Внешний цикл повторяется n раз. Если граф задан списками смежности, то внутренний цикл в строке 4 процедуры BFS для вершины x повторяется $\deg(x)$ раз, а всего $\sum_{x \in V} \deg(x) = 2m$ раз. Извлечение вершины из очереди, проверка ее новизны, а также обязательные действия, связанные с посещением вершины (помечивание ее как посещенной и помещение в очередь) выполняются за постоянное, не зависящее от параметров графа время. Таким образом, общая трудоемкость поиска в ширину оценивается как $O(m) + O(n) = O(m + n)$. Если же граф задан матрицей смежности, то для сканирования окрестности вершины (строка 4) необходимо полностью просмотреть соответствующую строку этой матрицы и общая трудоемкость будет $O(n^2)$.

Рассмотрим примеры применения поиска в ширину для решения конкретных задач.

Компоненты связности. Рассмотрим задачу выявления компонент связности графа. Ответ нужно получить в виде таблицы, в которой для каждой вершины x должен быть указан номер компоненты $comp(x)$, которой эта вершина принадлежит. Для решения этой задачи достаточно ввести переменную c со значением, равным текущему номеру компоненты, и каждый раз при посещении новой вершины x присваивать значение $comp(x) = c$. Значение c первоначально устанавливается равным 0 и увеличивается на 1 при каждом вызове процедуры BFS.

BFS-дерево. Допустим, поиск в ширину выполняется на связном графе. Каркас, образованный прямыми ребрами, можно рассматривать как корневое дерево с корнем в стартовой вершине. Оно называется *BFS-деревом*. В процессе обхода легко построить таблицу отцов F, задающую это дерево: достаточно при посещении вершины y при активной вершине x присвоить значение $F(y) = x$.

BFS-дерево с данным корнем не единственно, оно зависит от того, в каком порядке рассматриваются ребра, инцидентные активной вершине. Но все BFS-деревья обладают одним свойством, на котором основаны важнейшие применения поиска в ширину. Корневой каркас связного графа называется *деревом кратчайших путей*, если путь от любой вершины до корня в этом каркасе является кратчайшим путем между этими вершинами во всем графе.

Теорема 4.1 (о BFS-дереве). *Любое BFS-дерево является деревом кратчайших путей.*

Доказательство. Обозначим через $D(k)$ множество всех вершин графа, находящихся на расстоянии k от стартовой вершины a. Работа алгоритма начинается с посещения стартовой вершины, т.е. единственной вершины, составляющей множество $D(0)$. Далее будут посещены и помещены в очередь все вершины, смежные с a, т.е. вершины из множества $D(1)$. Затем эти вершины будут одна за другой извлекаться из очереди, становиться активными, и для каждой из них будут исследоваться все смежные вершины. Те из них, которые еще не посещались, будут посещены и помещены в очередь. Но это как раз все вершины из множества $D(2)$. Таким образом, все вершины из $D(2)$ посещаются позже всех вершин из $D(1)$. Ясно, что это продолжается и дальше: вершины из $D(k)$ посещаются позже вершин из $D(k-1)$, $k = 1,2, \dots$.

Докажем, что вершины из $D(k)$ будут в BFS-дереве находиться на расстоянии k от корня a. Для $k = 1$ это очевидно. Пусть $k > 1$ и $x \in D(k)$. Вершина x смежна с хотя бы одной вершиной из $D(k-1)$ и не смежна с вершинами из $D(i)$, $i < k-1$. Значит, она будет посещена, когда активной будет некоторая вершина $y \in D(k-1)$. При этом ребро (y, x) будет добавлено к дереву. Так как вершина y находится в дереве на расстоянии $k-1$ от корня (предположение индукции), то x будет на расстоянии k от корня. □

Таким образом, однократным выполнением поиска в ширину можно найти кратчайшие пути и расстояния от данной вершины a до всех остальных. Для этого нужно взять вершину a в качестве стартовой, положить $d(a, a) = 0$, а затем при каждом посещении новой вершины y при активной вершине x полагать $d(a, y) = d(a, x) + 1$.

Центр дерева. Центр дерева можно найти следующим образом. Выберем в данном дереве произвольную вершину a. Выполнив поиск в ширину из вершины a, найдем наиболее удаленную от нее вершину b. Выполнив второй раз поиск в ширину со стартом в вершине b, найдем наиболее удаленную от нее вершину c. Центр пути, соединяющего b и c, является центром всего дерева (см. задачу 4.23).

4.2. Поиск в глубину

При поиске в глубину в качестве активной выбирается та из открытых вершин, которая была посещена последней. Для реализации такого правила выбора наиболее удобной структурой хранения множества открытых вершин является стек: открываемые вершины складываются в стек в том порядке, в каком они посещаются, а в качестве активной выбирается последняя вершина.

Обозначим стек для открытых вершин через S, а верхний элемент стека – через $top(S)$. Предполагаем, что граф задан списками смежности; список вершин, смежных с вершиной x, обозначаем через $V(x)$. Операцию взятия очередной вершины y из списка $V(x)$ записываем как $y \Leftarrow V(x)$, при этом y удаляется из списка.

Процедура обхода одной компоненты связности методом поиска в глубину со стартовой вершиной a может быть записана следующим образом (DFS – от Depth First Search).

Procedure DFS(a)

```
1     посетить a;
2     while  S ≠ ∅  do
3           x := top(S);
4           if  V(x) ≠ ∅
5           then   y ⇐ V(x);
6                  if  y новая  then  посетить y
7           else   удалить x из  S
```

Здесь $x := top(S)$ означает, что переменной x присваивается значение, равное верхнему (последнему) элементу в стеке S, при этом сам этот элемент остается в стеке. При посещении вершина помечается как посещенная и помещается в стек.

При однократном выполнении процедуры DFS происходит обход компоненты связности, содержащей стартовую вершину a. Обход всего графа обеспечивается так же, как при поиске в ширину. Остается верной и оценка трудоемкости $O(m+n)$: ветвь **then** в строке 5 выполняется $2m$ раз, а ветвь **else** – n раз.

DFS-дерево. Поиск в глубину можно применить для нахождения компонент связности графа или для построения каркаса точно таким же образом, как поиск в ширину. Для связного графа каркас с корнем в стартовой вершине, получаемый поиском в глубину, называется *DFS-деревом*. Это дерево обладает особыми свойствами, на использовании которых основаны многие применения поиска в глубину. Рассмотрим наиболее важное из этих свойств.

Если в некотором связном графе выбрано корневое остовное дерево (каркас), то все обратные ребра, т.е. ребра графа, не принадлежащие дереву, можно разделить на две категории: ребро называется *продольным*, если одна из его вершин является предком другой в этом дереве, в противном случае оно называется *поперечным*.

В примере на рисунке 4.1 ребра каркаса выделены жирными линиями, корень – черным кружком. Обратные ребра показаны тонкими линиями, из них продольными являются ребра (1, 7), (2, 9), (3, 8), а поперечными – ребра (1, 2), (2, 5), (3, 5).

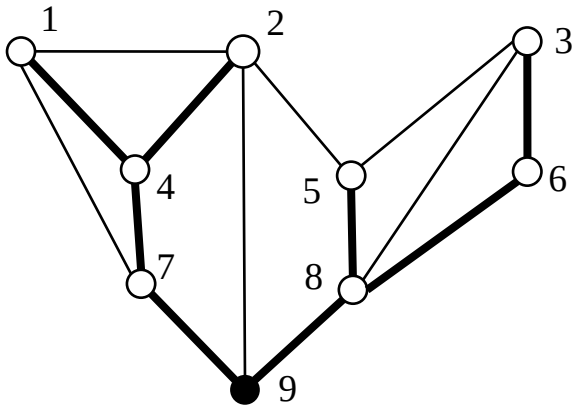


Рис. 4.1. Каркас графа и обратные ребра

Теорема 4.2 (о DFS-дереве). *Относительно DFS-дерева все обратные ребра являются продольными.*

Доказательство. Убедимся сначала, что после того, как стартовая вершина a помещена в стек, на каждом из последующих шагов работы алгоритма последовательность вершин, хранящаяся в стеке, образует путь с началом в вершине a, а все ребра этого пути принадлежат дереву. Вначале это, очевидно, так. В дальнейшем всякий раз, когда новая вершина y помещается в стек, к дереву добавляется прямое ребро (x, y), причем вершина x находится в стеке перед вершиной y. Значит, если указанное свойство имело место до добавления вершины y в стек, то оно сохранится и после добавления. Удаление же вершины из стека, конечно, не может нарушить этого свойства.

Пусть теперь (x, y) – обратное ребро. Допустим, вершина x посещается раньше, чем y. Рассмотрим шаг алгоритма, на котором посещается вершина y. При этом она помещается в стек. В этот момент вершина x еще находится в стеке. Действительно, вершина исключается из

стека только тогда, когда в ее окрестности нет новых вершин. Но непосредственно перед посещением вершина y является новой и принадлежит окрестности вершины x. Таким образом, вершина x лежит на пути, принадлежащем дереву и соединяющем вершины a и y. Но это означает, что вершина x является предком вершины y в дереве и, следовательно, ребро (x,y) – продольное. □

Таким образом, каркас, изображенный на рисунке 4.1, не мог быть построен методом поиска в глубину. Кстати, он не мог быть построен и с помощью поиска в ширину (почему?).

Рекурсивный вариант. Часто алгоритм поиска в глубину записывают в рекурсивной форме. Рекурсия позволяет представить алгоритм особенно компактно:

Procedure DFS(a)

1 посетить a;
2 **for** $x \in V(a)$ **do**
3 **if** x новая **then** DFS(x).

Предполагается, что вначале все вершины помечены как новые. Стек здесь скрыт в механизме рекурсии.

Шарниры и перешейки. Рассмотрим применение поиска в глубину к выявлению шарниров и перешейков графа. Напомним, что шарнир – это вершина, при удалении которой увеличивается число компонент связности, а перешеек – ребро с тем же свойством.

Допустим, требуется выяснить, является ли шарниром некоторая вершина a данного графа. Для решения этой задачи достаточно выполнить поиск в глубину со стартом в вершине a и построением DFS-дерева. Если теперь вершину a удалить из графа, то, ввиду отсутствия поперечных ребер, он распадется на столько компонент связности, сколько сыновей у вершины a в DFS-дереве. Значит, вершина a является шарниром тогда и только тогда, когда в DFS-дереве с корнем a степень корня больше 1.

Это свойство корня DFS-дерева можно было бы использовать для выявления всех шарниров, просто выполнив n раз поиск в глубину, стартуя поочередно в каждой вершине. Оказывается, все шарниры можно выявить, проделав поиск в глубину один раз. Первый алгоритм, который делает это, был предложен Тарьяном (R.E. Tarjan) в 1972 г. Здесь мы рассмотрим другой алгоритм, опубликованный в 2013 г. Шмидтом (J.M. Schmidt).

Предполагаем, что граф связный. Выполним поиск в глубину из произвольно выбранной вершины a с построением DFS-дерева. В процессе обхода графа будем нумеровать вершины в порядке их посещения (вершина a получает номер 1 и т.д.). Присвоенный таким образом вершине

x номер обозначим через $dn(x)$ и назовем ее *глубинным номером*. Очевидно, если вершина x является предком вершины y в DFS-дереве, то $dn(x) < dn(y)$. Каждое ребро DFS-дерева ориентируем по направлению к корню, т.е. от сына к отцу, а каждое обратное ребро – от предка к потомку. В примере на рисунке 4.2 показаны DFS-дерево (жирные ребра), глубинные номера вершин и ориентация ребер.

Теперь снова пометим все вершины графа как новые и будем их рассматривать в порядке возрастания глубинных номеров. Очередную (*активную*) вершину x пометим как посещенную (если она еще так не помечена) и рассмотрим все выходящие из нее обратные ребра. Для каждого такого ребра найдем ориентированный путь, начинающийся этим ребром и заканчивающийся первой после x на этом пути посещенной вершиной (этот путь единственный). Путь может закончиться в вершине x, т.е. оказаться циклом. В этом случае скажем, что вершина x – начало цикла. Каждое ребро пути пометим как пройденное, а каждую вершину – как посещенную. Все полученное таким образом множество путей будем называть *путевым разложением* графа.

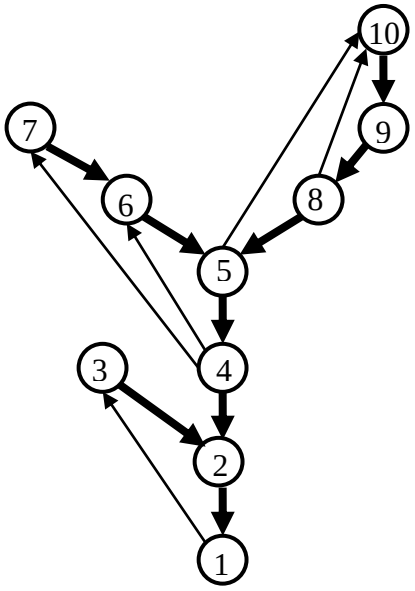


Рис. 4.2. DFS-дерево, глубинные номера и ориентация ребер

В следующей таблице показаны этапы работы этого алгоритма на графе с рисунка 4.2:

Ребро	Путь	Посещенные вершины
(1,3)	(1,3,2,1)	1,2,3
(4,6)	(4,6,5,4)	1,2,3,4.5,6
(4,7)	(4,7,6)	1,2,3,4.5,6,7
(5,10)	(5,10,9,8,5)	1,2,3,4.5,6,7,8,9,10
(8,10)	(8,10)	

Началами циклов, как видно, здесь являются вершины 1, 4, 5, и по окончании работы все ребра помечены как пройденные, кроме ребра (2,4). Следующие два утверждения показывают, как по этим данным найти перешейки и шарниры графа.

Утверждение 1. *Ребро является перешейком тогда и только тогда, когда оно не принадлежит ни одному пути из путевого разложения.*

Доказательство. Ранее было доказано (теорема 1.4), что ребро является перешейком тогда и только тогда, когда оно не цикловое, т.е. через него не проходит ни один цикл. Остается показать, что ребро будет пройдено тогда и только тогда, когда оно цикловое. Всякое обратное ребро относительно DFS-дерева образует цикл с ребрами этого дерева и путь, начинающийся этим ребром, который строится в ходе работы алгоритма, – часть этого цикла. Значит, всякое пройденное ребро является цикловым. Обратно, пусть (x, y) – цикловое ребро. Если это ребро обратное, то оно будет пройдено, т.к. для каждого обратного ребра алгоритм строит путь, начинающийся этим ребром. Допустим, это ребро прямое, т.е. принадлежит дереву, причем $dn(x) < dn(y)$. Так как оно цикловое, то существует охватывающее его обратное ребро, т.е. ребро, соединяющее предка вершины x с потомком вершины y. Возьмем первое из таких ребер, рассматриваемых в ходе работы алгоритма. Когда строится путь, начинающийся этим ребром, вершина y и ее потомки еще не посещены, поэтому этот путь пройдет через ребро (x, y). □

Очевидно, каждая вершина, инцидентная перешейку и имеющая степень больше 1, является шарниром. Остальные шарниры можно найти, опираясь на следующий факт.

Утверждение 2. *Вершина, отличная от корня DFS-дерева и не инцидентная перешейку, является шарниром тогда и только тогда, когда она – начало некоторого цикла в путевом разложении.*

Доказательство. Пусть x – начало цикла C, $x \neq a$, y – предпоследняя вершина этого цикла (первой и последней является x). Тогда нет обратных ребер, соединяющих собственных предков вершины x с потомками вершины y (при наличии такого ребра вершина y была бы уже посещена в тот момент, когда активной становится вершина x, и вместо цикла C получился бы путь, не являющийся циклом). Значит, если удалить из графа вершину x, то в оставшемся графе не будет ребер, соединяющих вершины из ветви с корнем y с остальной частью графа, т.е. граф станет несвязным. Следовательно, в этом случае вершина x – шарнир.

Обратно, пусть x – шарнир, не принадлежащий никакому перешейку. При удалении вершины x граф распадается на несколько компонент связности, одна из них содержит корень a, все остальные порождаются

множествами вершин некоторых ветвей DFS-дерева в сыновьях вершины x. Пусть T_y – одна из таких «отделившихся» ветвей с корнем в вершине y. Тогда в графе нет обратных ребер, соединяющих собственных предков вершины x с потомками вершины y. Поэтому в тот момент, когда в ходе работы алгоритма активной станет вершина x, вершина y и все ее потомки еще не посещены. Так как ребро (x, y) – не перешеек, то есть хотя бы одно обратное ребро, соединяющее вершину x с потомком вершины y. Путь, начинающийся первым из таких ребер, и будет циклом с началом в вершине x. □

Блоки. *Блок* графа – это его максимальный связный подграф, не имеющий собственных шарниров (т.е. некоторые шарниры графа могут принадлежать блоку, но своих шарниров у блока нет). Каждое ребро графа принадлежит в точности одному блоку, а вершина может принадлежать нескольким блокам, но только в том случае, когда она – шарнир. Строение связного графа G, состоящего из нескольких блоков, описывается *BC-деревом*, которое строится следующим образом. В этом дереве вершины двух типов – одни соответствуют блокам графа G (вершины-блоки), другие – его шарнирам (вершины-шарниры). Вершина-блок соединяется в BC-дереве ребром с вершиной-шарниром, если соответствующий шарнир принадлежит соответствующему блоку.

Путевое разложение графа, описанное выше, можно использовать и для выявления блоков (см. задачу 4.25).

Задачи

4.1. Используя метод поиска в ширину, найдите компоненты связности графа, заданного матрицей смежности:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

4.2. Методом поиска в ширину постройте дерево кратчайших путей из вершины 1 для графа, заданного списками смежности:
1: 2,9; 2: 1, 3, 7, 8; 3: 2, 7; 4: 5, 9; 5: 4, 6, 7, 9, 10;
6: 5, 7, 10; 7: 2, 3, 5, 6, 8; 8: 2, 7, 9; 9: 1, 4. 5,8,10; 10: 5, 6, 9.

4.3. Лес задан списками смежности. Какие из следующих оценок времени работы верны для поиска в ширину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.4. Лес задан матрицей смежности. Какие из следующих оценок времени работы верны для поиска в ширину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.5. Какова будет высота BFS-дерева, если поиск в ширину применяется к графу 1) K_n; 2) $K_{p,q}$, $p,q>1$; 3) Q_k; 4) C_n; 5) $P_k \times P_l$, старт в вершине степени 2?

4.6. Пусть h –высота BFS-дерева для графа G. Может быть $h > \operatorname{diam} G$? $h < \operatorname{rad} G$? Всегда ли можно выбрать стартовую вершину так, чтобы было $h = \operatorname{diam} G$? $h = \operatorname{rad} G$?

4.7. Ребро (x,y) является обратным ребром относительно BFS-дерева с корнем a. Какие из следующих соотношений могут выполняться? А если граф двудольный?
1) $d(a,x) = d(a,y)$; 3) $|d(a,x) - d(a,y)| = 2$;
2) $|d(a,x) - d(a,y)| = 1$; 4) $|d(a,x) - d(a,y)| = 3$.

4.8. Два человека имеют восьмилитровый кувшин, наполненный вином, и два пустых кувшина: а) трехлитровый и двухлитровый;
б) пятилитровый и шестилитровый.
Они могут переливать вино из одного кувшина в другой, пока либо первый не опустеет, либо второй не наполнится. Могут они разделить вино поровну с помощью таких действий? Какое наименьшее число переливаний потребуется?

4.9. Используя алгоритм поиска в глубину, найдите компоненты связности графа, заданного списками смежности:
1: 2, 4, 6; 2: 1; 3: 5, 9; 4: 1; 5: 3, 9; 6: 1; 7: 8, 10;
8: 7, 10, 11; 9: 3, 5; 10: 7, 8, 12; 11: 8,12; 12: 10, 11.

4.10. Постройте DFS-дерево с корнем в вершине 1 для графа, заданного списками смежности:
1: 5, 6, 7, 8; 2: 3, 4, 7, 8; 3: 2, 4; 4: 2, 3, 7; 5: 1, 6, 10;
6: 1, 5; 7: 1, 2, 4, 8; 8: 1, 2, 7, 9; 9: 8; 10: 5.

4.11. Планарный граф задан списками смежности. Какие из следующих оценок времени работы верны для поиска в глубину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.12. Планарный граф задан матрицей смежности. Какие из следующих оценок времени работы верны для поиска в глубину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.13. Дерево задано списками смежности. Какие из следующих оценок времени работы верны для поиска в глубину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.14. Сколько различных DFS-деревьев можно построить для графа 1) K_n; 2) P_n; 3) C_n?

4.15. Пусть h –высота DFS-дерева для графа G. Может быть
а) $h > \operatorname{diam} G$?
б) $h < \operatorname{rad} G$?

4.16. Какой может быть максимальная высота DFS-дерева, если поиск в глубину применяется к графу 1) $P_k \times P_l$; 2) Q_k; 3) $K_{p,q}$, $p > q$?

4.17. Ребро (x, y) является обратным ребром относительно DFS-дерева с корнем a. Какие из следующих соотношений могут выполняться (d обозначает расстояние между вершинами в дереве)? А если граф двудольный?
1) $d(a,x) = d(a,y)$; 3) $|d(a,x) - d(a,y)| = 2$;
2) $|d(a,x) - d(a,y)| = 1$; 4) $|d(a,x) - d(a,y)| = 3$.

4.18. Постройте DFS-дерево и путевое разложение для графа, заданного списками смежности:
1: 3, 4; 2: 5; 3: 1, 6; 4: 1, 6; 5: 2, 7, 11; 6: 3, 4, 7, 8, 9;
7: 5, 6, 10, 11; 8: 6, 9; 9: 6, 8; 10: 7; 11: 5,7,12; 12: 11.
Найдите все шарниры и блоки этого графа, постройте BC- дерево.

4.19. Каждый блок графа состоит из трех вершин. Сколько вершин и ребер в этом графе, если его BC-дерево представляет собой а) путь P_{21}; б) путь P_{100}; в) граф $K_{1,20}$?

4.20*. Разработайте алгоритм, который за время $O(n)$ проверяет, является ли данный граф деревом.

4.21*. Разработайте алгоритм, проверяющий, является ли данный граф двудольным.

4.22*. Разработайте алгоритм, который находит кратчайший цикл, проходящий через данную вершину.

4.23*. (Задача об одностороннем движении) Докажите, что ребра обыкновенного графа можно ориентировать так, что получится сильно связный орграф, тогда и только тогда, когда в нем нет перешейков. Разработайте алгоритм построения такой ориентации.

4.24**. Разработайте алгоритм выявления всех блоков графа по известному путевому разложению.

4.25**. Докажите, что описанный выше алгоритм нахождения центра дерева с помощью двойного обхода в ширину действительно находит центр любого дерева. Приведите пример графа, для которого это не так.

5. Циклы

5.1. Эйлеровы циклы

Эйлеровым циклом (*путем*) называется цикл (путь), проходящий через все ребра графа. Заметим, что по определению путь или цикл не может дважды проходить по одному ребру, поэтому эйлеров путь или цикл проходит по каждому ребру графа в точности один раз.

Теорема 5.1 (об эйлеровом цикле). *Эйлеров цикл в связном графе существует тогда и только тогда, когда в нем степени всех вершин четны.*

Доказательство. Пусть G – связный граф с четными степенями всех вершин и пусть $P = (x_1, x_2, \dots, x_k)$ – путь наибольшей длины в этом графе. Покажем, что этот путь является циклом и что он проходит через все ребра графа, т.е. P – эйлеров цикл.

Действительно, предположим, что P – не цикл, т.е. $x_k \neq x_1$. Путь P проходит через все ребра, инцидентные вершине x_k (иначе можно было бы построить более длинный путь, добавив в конце пути P еще одно ребро). Допустим, что через вершину x_k этот путь проходит s раз. При этом $s-1$ раз при прохождении через эту вершину используются два ребра, а последний раз – одно ребро. Получается, что степень вершины x_k равна $2(s-1)+1$, а это противоречит условию четности степеней.

Допустим, что P проходит не через все ребра графа. Так как граф связен, то среди ребер, не принадлежащих P, имеется ребро, инцидентное вершине из P. Пусть (x_i, y) – такое ребро. Тогда можно построить более длинный путь: $x_i, x_{i+1}, \dots, x_k, x_1, x_2, \dots, x_i, y$. □

Следствие (об эйлеровом пути). *Эйлеров путь в связном графе существует тогда и только тогда, когда в нем имеется не более двух вершин нечетной степени.*

Доказательство. Если в графе нет вершин нечетной степени, то в нем имеется эйлеров цикл, он является и эйлеровым путем. Не существует графов с одной вершиной нечетной степени, так как по теореме о рукопожатиях сумма всех степеней в любом графе – четное число. Допустим в графе G есть ровно две вершины нечетной степени, a и b. Построим новый граф H, добавив к графу G новую вершину c и ребра (a, c), (b, c). В графе H степени всех вершин четны, значит, в нем есть эйлеров цикл. Пусть $x_1, x_2, \dots, x_k, x_1$ – такой цикл. Обход по циклу можно начинать в любой вершине, поэтому можно считать, что $x_1 = c$. Но тогда $x_2, x_3, \dots, x_k$ – эйлеров путь в графе G. □

Рассмотрим алгоритм построения эйлерова цикла. Предполагается, что условия существования уже проверены – граф связен и степени четны. В алгоритме строится путь, начинающийся в произвольно выбранной стартовой вершине a, при этом каждый раз для дальнейшего продвижения выбирается любое еще не пройденное ребро. Вершины пути накапливаются в стеке S (они могут повторяться). Когда наступает тупик – все ребра, инцидентные последней вершине пути, уже пройдены, производится возвращение вдоль пройденного пути тех пор, пока не встретится вершина, которой инцидентно еще не пройденное ребро. При возвращении вершины перекладываются из стека S в другой стек C. Затем возобновляется движение вперед по не пройденным ребрам, пока снова не наступает тупик, и т.д. Процесс заканчивается, когда стек S оказывается пустым. В этот момент в стеке C находится последовательность вершин эйлерова цикла.

Построение эйлерова цикла

```
1       выбрать произвольно вершину a;
2       поместить a в S;
3       while S ≠ ∅ do
4           x := top(S);
5           if имеется не пройденное ребро (x, y)
6           then пометить ребро (x, y) как пройденное;
7                поместить y в S;
8           else переместить вершину x из S в C
```

Для обоснования алгоритма заметим, что в конечном итоге все ребра будут пройдены – это следует из связности графа. Работа алгоритма состоит из серий поступательных движений, когда несколько раз подряд выполняется ветвь **then** (строка 6), и серий возвратных движений, когда повторяется ветвь **else**. Из-за четности степеней каждая поступательная серия заканчивается в той вершине, в которой начиналась. Поэтому каждая следующая возвратная серия начинается в той вершине, в которой закончилась предыдущая. Таким образом, последовательность вершин, накапливаемая в стеке C, представляет собой путь, состоящий из пройденных ребер. Так как в момент окончания работы стек S пуст, то этот путь в итоге содержит все ребра графа.

Время работы этого алгоритма оценивается как $O(m)$. Этот вывод справедлив лишь при определенных предположениях о том, как задан граф. Можно, например, использовать две структуры:

- массив ребер, в котором в позиции, соответствующей ребру, помещается запись, указывающая две вершины этого ребра и пометку о том, пройдено ли это ребро;

- списки инцидентности, в которых для каждой вершины перечисляются инцидентные ей ребра.

В ориентированном графе эйлеров цикл – это ориентированный цикл, проходящий через все ребра. Приведем без доказательства критерий существования эйлерова цикла в орграфе.

Теорема 5.2. *Эйлеров цикл в сильно связном орграфе существует тогда и только тогда, когда в нем для каждой вершины* x *выполняется равенство* $deg^+(x) = deg^-(x)$.

Последовательности де Брёйна. Одно из применений эйлеровых циклов в орграфе – построение так называемых последовательностей де Брёйна (de Bruijn). Последовательностью де Брёйна порядка k называется такое слово $a_1a_2 \dots a_n$ в алфавите $\{0,1\}$, что в слове $a_1a_2 \dots a_na_1a_2 \dots a_{k-1}$ среди отрезков длины k каждое слово длины k встречается ровно один раз. Иначе говоря, если последовательность де Брёйна свернуть в кольцо и двигать вдоль этого кольца окно, в котором видны k последовательных букв, то в течение одного оборота каждое слово длины k появится в окне один раз. Ясно, что $n = 2^k$. Пример последовательности де Брейна порядка 3: 00010111.

Для построения последовательностей де Брёйна можно использовать граф де Брёйна. Вершинами этого графа являются всевозможные слова длины $k-1$. Из каждой вершины $x_1x_2 \dots x_{k-1}$ выходят ровно два ребра: в вершины $x_2x_3 \dots x_{k-1}0$ и $x_2x_3 \dots x_{k-1}1$. Первому ребру приписывается буква 0, второму – буква 1. Очевидно, входящих ребер тоже будет два: из вершин $0x_1 \dots x_{k-2}$ и $1x_1 \dots x_{k-2}$. Очевидно также, что этот граф сильно связен – от любого слова можно перейти к любому другому, добавляя буквы в конце. Значит, в нем имеется эйлеров цикл. Двигаясь вдоль такого цикла и выписывая буквы, приписанные ребрам, получим последовательность де Брёйна. На рисунке 5.1 показан граф де Брейна для $k = 3$.

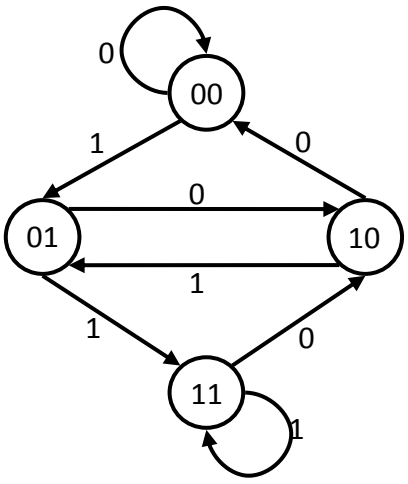


Рис. 5.1. Граф де Брейна для $k = 3$

5.2. Гамильтоновы циклы

Гамильтоновым циклом (*путем*) называют простой цикл (путь), содержащий все вершины графа.

Внешне определение гамильтонова цикла похоже на определение эйлерова цикла. Однако есть кардинальное различие в сложности решения соответствующих задач. Мы видели, что имеется достаточно простой критерий существования эйлерова цикла и быстродействующий алгоритм его построения. Для гамильтоновых же циклов (и путей) неизвестно никаких легко проверяемых необходимых и достаточных условий их существования, а все известные алгоритмы требуют для некоторых графов перебора большого числа вариантов. Такие задачи называют задачами переборного типа или неподдающимися задачами. Очень многие известные задачи относятся к разряду неподдающихся, среди них немало задач на графах. Существует математическая теория сложности алгоритмов и задач, в которой под эффективным алгоритмом понимают алгоритм, время работы которого ограничено сверху полиномом от длины записи входных данных. Выводы этой теории делают весьма правдоподобным предположение о том, что для многих неподдающихся задач, в том числе и для задачи о гамильтоновом цикле, не существует эффективных алгоритмов.

Перебор вариантов в задаче о гамильтоновом цикле (построить гамильтонов цикл или убедиться, что его не существует) можно организовать с помощью дерева путей. Это дерево представляет всевозможные простые пути в данном графе, начинающиеся в некоторой (произвольно выбранной) вершине a. Его вершины соответствуют вершинам графа, при этом каждая вершина графа может быть представлена в дереве несколько раз. Оно может быть построено следующим образом. Выберем в графе вершину a и объявим ее корнем дерева. Вершины, смежные с a, добавим к дереву в качестве сыновей вершины a. Для каждой вершины b, добавленной к дереву, рассмотрим все смежные с ней вершины и те из них, которые не лежат на пути (в дереве) из a в b, добавим к дереву в качестве сыновей вершины b. Процесс построения дерева заканчивается, когда к нему уже невозможно добавить новую вершину. Очевидно, что каждому пути в дереве, начинающемся в корне, соответствует точно такой же (простой) путь в графе, и обратно, каждому простому пути в графе с началом в вершине a соответствует такой же путь в дереве. Каждой вершине, находящейся в дереве на расстоянии $n-1$ от корня, соответствует гамильтонов путь в графе с началом в вершине a, а если последняя вершина этого пути смежна с a, то получаем гамильтонов цикл. Если высота дерева путей не превосходит $n-2$, то в графе нет гамильтоновых путей, начинающихся в a, и нет гамильтоновых циклов. На рисунке 5.2 показаны граф и его дерево путей из вершины 1.

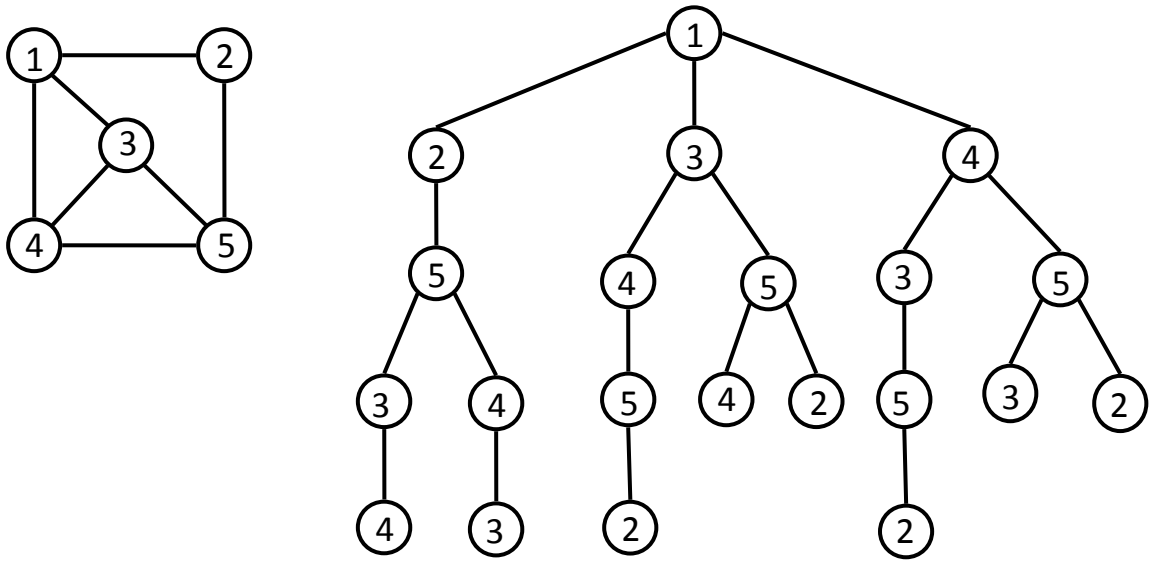


Рис. 5.2. Граф и его дерево путей

Дерево путей можно построить с помощью процедуры типа поиска в ширину: вершины исследуются и добавляются к дереву в порядке возрастания их расстояний (в дереве) от корня. Отличие от обычного поиска в ширину состоит в том, что вершина графа может добавляться к дереву несколько раз.

Дерево путей может быть очень большим. Для полного графа K_n в этом дереве будет $(n-1)!$ листьев. Но если задача состоит в том, чтобы найти один гамильтонов путь или цикл (если такой существует), то совсем не обязательно строить это дерево целиком. Можно организовать обход этого дерева с помощью, скажем, поиска в глубину, без его явного построения. Для полного графа такой алгоритм даст ответ очень быстро, но в некоторых случаях время его работы тоже растет с факториальной скоростью. Например, для графа $K_{n-1}+K_1$, если в качестве стартовой выбрана не изолированная вершина, будут рассмотрены все $(n-2)!$ простых путей длины $n-2$ в большой компоненте.

Код Грея. Кодом Грея порядка k называется расположение всех 2^k двоичных слов длины k в последовательность, в которой любые два соседних слова различаются ровно в одной букве. Например, при $k=3$ следующая последовательность образует код Грея:

000, 001, 011, 010, 110, 111, 101, 100.

Двоичные слова длины k можно рассматривать как вершины графа k-мерного куба Q_k – ребра этого графа соединяют как раз пары слов, различающихся в одной букве. Значит, код Грея есть не что иное, как гамильтонов путь в графе Q_k. Покажем, что такой путь существует при любом k. Куб Q_k можно разделить на два куба размерности $k-1$: один, $Q_k(0)$, состоит из вершин $x_1x_2\ldots x_n$, у которых $x_n=0$, другой, $Q_k(1)$, – из

тех, у которых $x_n = 1$. При этом вершина $x_1x_2 \dots x_{n-1}0$ смежна с вершиной $x_1x_2 \dots x_{n-1}1$. Если известен гамильтонов путь в графе Q_{k-1}, то гамильтонов путь в графе Q_k можно построить так: строим гамильтонов путь в подграфе $Q_k(0)$, из последней вершины переходим в граф $Q_k(1)$ и в нем воспроизводим тот же гамильтонов путь, но в обратном порядке.

Гамильтоновы пути в турнирах. В ориентированном графе гамильтонов цикл (путь) – это ориентированный цикл (путь), проходящий через каждую вершину точно один раз. Задачи о гамильтоновых циклах и путях для орграфов в общем случае тоже неподдающиеся. В то же время имеется важный класс графов, для которых эти задачи имеют простое решение. Это так называемые турниры.

Орграф называется *турниром*, если для каждых двух вершин в нем имеется единственное ребро, соединяющее эти вершины.

Теорема 5.3. *В любом турнире имеется гамильтонов путь.*

Доказательство. Пусть G – турнир и $P = (x_1, x_2, \dots, x_k)$ – самый длинный простой ориентированный путь в G. Докажем, что каждая вершина графа принадлежит этому пути. Допустим, путь P не проходит через вершину y. Граф G содержит одно из ребер (x_1, y), (y, x_1). Но он не может содержать ребро (y, x_1), так как иначе получился бы более длинный путь $y, x_1, x_2, \dots, x_k$. Значит, он содержит ребро (x_1, y). Аналогично убеждаемся, что он содержит ребро (y, x_k). Пусть i – наименьшее, при котором ребро (y, x_i) принадлежит графу. Тогда последовательность $x_1, \dots, x_{i-1}, y, x_i, \dots, x_k$ образует ориентированный путь большей длины.

5.2. Пространство циклов

Циклы – очень важная часть структуры графа, с ними приходится иметь дело при решении многих задач. В графе может быть очень много циклов (см. задачу 5.15), поэтому полезно иметь средства для компактного описания множества всех циклов данного графа и манипулирования ими. Одно из наиболее удобных средств такого рода предоставляет аппарат линейной алгебры.

Пространство подграфов. Зафиксируем некоторое множество V и рассмотрим множество Γ_V всех графов с множеством вершин V. Буквой O будем обозначать пустой граф из этого множества: $O = (V, \emptyset)$.

Для графов $G_1 = (V, E_1)$, и $G_2 = (V, E_2)$ определим их *сумму по модулю 2* (в дальнейшем в этой главе будем называть ее просто суммой) как граф $G_1 \oplus G_2 = (V, E_1 \otimes E_2)$, где $E_1 \otimes E_2$ обозначает симметрическую разность множеств. Иначе говоря, ребро принадлежит графу $G_1 \oplus G_2$ тогда

и только тогда, когда оно принадлежит в точности одному из графов G_1 и G_2. Следующие свойства этой операции очевидны или легко проверяются.

1) Коммутативность: $G_1 \oplus G_2 = G_2 \oplus G_1$ для любых G_1 и G_2.
2) Ассоциативность: $G_1 \oplus (G_2 \oplus G_3) = (G_1 \oplus G_2) \oplus G_3$ для любых G_1, G_2, G_3.
3) $G \oplus O = G$ для любого G.
4) $G \oplus G = O$ для любого G.

Отсюда следует, что множество Γ_V относительно сложения по модулю 2 образует абелеву группу. Нейтральным элементом («нулем») этой группы служит граф O, а противоположным к каждому графу является сам этот граф. Уравнение $G \oplus X = H$ с неизвестным графом X и заданными графами G и H имеет единственное решение $X = G \oplus H$. Благодаря свойству ассоциативности мы можем образовывать выражения вида $G_1 \oplus G_2 \oplus \ldots \oplus G_k$, не используя скобок для указания порядка действий. Ребро принадлежит графу $G_1 \oplus G_2 \oplus \ldots \oplus G_k$ тогда и только тогда, когда оно принадлежит нечетному количеству графов-слагаемых.

Рассмотрим множество из двух элементов $\{0,1\}$. Оно является полем относительно операций умножения и сложения по модулю 2. Определим операцию умножения элементов этого поля на графы: $0 \cdot G = O$, $1 \cdot G = G$ для любого графа G. Теперь можно образовывать линейные комбинации графов вида $\alpha_1 G_1 \oplus G_2 \oplus \ldots \oplus \alpha_k G_k$ с коэффициентами $\alpha_i \in \{0,1\}$. Каждая такая комбинация тоже принадлежит Γ_V. Тем самым Γ_V становится линейным векторным пространством над полем из двух элементов.

Зафиксируем некоторый граф $G = (V, E)$ и рассмотрим множество всех его остовных подграфов. Это множество состоит из 2^m элементов, где m – число ребер у графа G. Оно замкнуто относительно сложения графов и умножения на элементы поля, следовательно, является подпространством пространства Γ_V. Его называют *пространством подграфов* графа G.

Любой граф из пространства подграфов может быть выражен как сумма однореберных подграфов. Всего у графа G имеется m однореберных подграфов и они, очевидно, линейно независимы. Следовательно, однореберные подграфы образуют базис пространства подграфов, а размерность этого пространства равна m.

В пространстве подграфов можно естественным способом ввести координаты. Пронумеруем ребра графа G: $E = \{e_1, e_2, \ldots, e_m\}$. Теперь остовному подграфу H можно поставить в соответствие характеристический вектор $\eta(H) = (\eta_1, \eta_2, \ldots, \eta_m)$ его множества ребер:

$$\eta_i = \begin{cases} 0, & \text{если ребро } e_i \text{ не принадлежит } H, \\ 1, & \text{если ребро } e_i \text{ принадлежит } H. \end{cases}$$

Получаем взаимно однозначное соответствие между множеством остовных подграфов и множеством всех m–мерных двоичных векторов. Сумме графов соответствует векторная (покоординатная) сумма по модулю 2 их характеристических векторов.

Квазициклы. Далее в этой главе *циклом графа* G будем называть его остовный подграф, у которого одна компонента связности является простым циклом, а остальные – изолированными вершинами. На рисунке 5.3 показано, что в результате сложения двух циклов может получиться цикл. Это не всегда так (например, если складываемые циклы не имеют общих ребер), но все же графы, которые являются суммами циклов, обладают некоторыми особенностями. На этом основан алгебраический подход к описанию и изучению множества циклов графа.

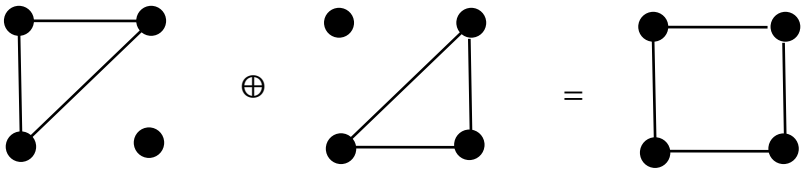


Рис. 5.3. Сложение по модулю 2 двух циклов

Остовный подграф графа G, у которого степени всех вершин четны, называется *квазициклом*. Таким образом, любой цикл является квазициклом и граф O – квазицикл.

Лемма 1. *Любой непустой квазицикл является суммой циклов.*

Доказательство. Пусть H – квазицикл, $H \neq O$. Возьмем какую-нибудь компоненту связности в H, содержащую более одной вершины. Степени всех вершин в этой компоненте не меньше 2 и по теореме 1.2 в нем есть цикл. Взяв какой-нибудь простой цикл в этой компоненте и добавив к нему изолированные вершины, получим цикл графа G, обозначим его C_1. Удалим ребра подграфа C_1 из H, получим подграф H_1, который тоже является квазициклом. Если в H_1 есть хотя бы одно ребро, то в нем аналогичным образом можно найти цикл C_2. Удалив его ребра из H_1, получим квазицикл H_2, и т.д. В конце концов, когда останется пустой граф, будут найдены циклы $C_1, C_2, \ldots, C_k$, попарно не имеющие общих ребер и в совокупности содержащие все ребра графа H. Таким образом, $H = C_1 \oplus C_2 \oplus \ldots \oplus C_k$. □

Лемма 2. *Сумма любых двух квазициклов есть квазицикл.*

Доказательство. Пусть H_1 и H_2 – квазициклы графа G. Рассмотрим произвольную вершину $a \in V$, и пусть ее степени в H_1 и H_2 равны соответственно d_1 и d_2. Тогда степень вершины a в графе $H_1 \oplus H_2$ будет равна $d = d_1 + d_2 - 2d_0$, где d_0 – число вершин, с которыми вершина a

смежна в обоих графах H_1 и H_2. Отсюда видно, что число d четно, если четны оба числа d_1 и d_2. □

Из двух доказанных лемм следует справедливость следующей теоремы.

Теорема 5.4 (о квазициклах). *Граф является квазициклом тогда и только тогда, когда он может быть представлен как сумма циклов.*

Таким образом, множество всех квазициклов графа G замкнуто относительно сложения по модулю 2. Очевидно, оно также замкнуто относительно умножения на 0 и 1. Значит, оно образует линейное подпространство пространства подграфов. Его называют *пространством циклов* графа G.

Фундаментальные циклы. Компактное представление пространства дает его базис. Если выписать все простые циклы графа G, то это в большинстве случаев не будет его базисом, так как некоторые из этих циклов могут быть суммами других. Построить базис пространства циклов, состоящий из простых циклов, можно следующим образом. Выберем в графе G какой-нибудь каркас T. Пусть $e_1, e_2, \dots, e_s$ – все ребра графа, не принадлежащие T. Каждое ребро e_iвместе с ребрами каркаса T образует цикл Z_i. Таким образом, получаем семейство из s циклов, они называются *фундаментальными циклами* относительно каркаса T.

Теорема 5.5 (о фундаментальных циклах). *Множество всех фундаментальных циклов относительно любого каркаса T графа G образует базис пространства циклов этого графа.*

Доказательство. Зафиксируем некоторый каркас T и рассмотрим фундаментальные циклы $Z_1, Z_2, \dots, Z_s$ относительно этого каркаса. В каждом цикле Z_i имеется ребро e_i, принадлежащее этому циклу и не принадлежащее никакому из остальных. Поэтому при сложении этого цикла с другими фундаментальными циклами данное ребро не «уничтожится» и будет присутствовать в суммарном графе. Следовательно, сумма различных фундаментальных циклов никогда не будет пустым графом, то есть фундаментальные циклы линейно независимы.

Покажем теперь, что любой квазицикл графа G является суммой фундаментальных циклов. Действительно, пусть H – квазицикл. Пусть $e_{i_1}, e_{i_2}, \dots, e_{i_t}$ – все ребра подграфа H, не принадлежащие каркасу T. Рассмотрим граф $F = H \oplus Z_{i_1} \oplus Z_{i_2} \oplus \dots \oplus Z_{i_t}$. Каждое из ребер e_{i_j}, $j = 1,2, \dots t$, входит ровно в два слагаемых этой суммы – в H и в Z_{i_j}. Следовательно, при сложении все эти ребра уничтожатся. Все остальные ребра, присутствующие в графах–слагаемых, принадлежат каркасу T. Значит, граф F – подграф графа T. Так как все слагаемые являются

квазициклами, то F – квазицикл. Но в T нет циклов, поэтому имеется единственная возможность: $F = O$, откуда получаем $H = Z_{i_1} \oplus Z_{i_2} \oplus \ldots \oplus Z_{i_t}$. □

Из этой теоремы следует, что размерность пространства циклов графа равна числу ребер, не входящих в его каркас. Так как каждый каркас содержит $n - k$ ребер, где k – число компонент связности графа, то число ребер, не входящих в каркас, равно $m - n + k$. Это и есть размерность пространства циклов, это число обозначается через $\nu(G)$ и называется *цикломатическим числом* графа.

Построение базиса циклов. Базис пространства циклов графа коротко называют *базисом циклов*. На основании теоремы 5.5 можно предложить простой способ построения базы циклов графа. Сначала находится какой-нибудь каркас, затем для каждого ребра, не принадлежащего каркасу, отыскивается тот единственный цикл, который это ребро образует с ребрами каркаса. Таким образом, любой алгоритм построения каркаса может быть использован для нахождения базы циклов.

Поиск в глубину особенно удобен благодаря основному свойству DFS-дерева – каждое обратное ребро относительно этого дерева является продольным. Это означает, что из двух вершин такого ребра одна является предком другой в DFS-дереве. Каждое такое ребро в процессе поиска в глубину встретится дважды – один раз, когда активной вершиной будет предок, другой раз, когда ею будет потомок. В этом последнем случае искомый фундаментальный цикл состоит из рассматриваемого обратного ребра и участка пути в DFS-дереве, соединяющего эти две вершины. Но этот путь так или иначе запоминается в процессе обхода в глубину, так как он необходим для последующего возвращения. Если, например, для хранения открытых вершин используется стек, как было описано в предыдущей главе, то вершины этого пути находятся в верхней части стека. В любом случае этот путь легко доступен и цикл находится без труда.

Хотя сам поиск в глубину выполняется за линейное от числа вершин и ребер время, решающее влияние на трудоемкость этого алгоритма оказывает необходимость запоминать встречающиеся циклы. Подсчитаем суммарную длину этих циклов для полного графа с n вершинами. DFS-дерево в этом случае является простым путем, относительно него будет $n - 2$ цикла длины 3, $n - 3$ цикла длины 4, ... , 1 цикл длины n. Сумма длин всех фундаментальных циклов будет равна

$$\sum_{i=1}^{n-2} i(n + 1 - i) = \frac{n^3 + 3n^2 - 16n + 12}{6} = O(n^3).$$

Таким образом, для некоторых графов число операций этого алгоритма будет величиной порядка n^3.

Базис циклов плоского графа. Не всякий базис циклов является системой фундаментальных циклов относительно некоторого каркаса. Например, у связного плоского графа, не имеющего шарниров, граница каждой грани является простым циклом. Можно доказать, что множество всех циклов, ограничивающих внутренние грани, образует базис циклов такого графа. Например, у графа, показанного на рисунке 5.4, четыре треугольника составляют базис циклов. Этот базис не является системой фундаментальных циклов относительно какого-либо каркаса. Действительно, такой каркас обязательно должен содержать два ребра внутреннего треугольника, например, ребра (2,3) и (2,5). Тогда он должен содержать и одно из ребер (3,6) или (5,6). В любом случае цикл (2,3,6,5,2) будет одним из фундаментальных циклов.

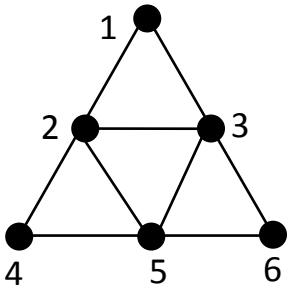


Рис. 5.4. Плоский граф

Пространство разрезов. С пространством циклов тесно связано *пространство разрезов* графа. Пусть $A \subseteq V$, $\bar{A} = V - A$. *Разрез* графа G, определяемый множеством A, – это остовный подграф, ребрами которого являются все ребра графа, соединяющие вершины из A с вершинами из $\bar{A}$. Этот подграф будем обозначать через $S(A)$. Если $A = \emptyset$, то $S(A) = O$. Очевидно, $S(A) = S(\bar{A})$.

Теорема 5.6 (о разрезах). *Множество всех разрезов данного графа замкнуто относительно сложения по модулю 2.*

Доказательство. Пусть $A \subseteq V$, $B \subseteq V$. Рассмотрим граф $S(A) \oplus S(B)$. Диаграмма на рисунке 5.5 помогает понять, какие ребра принадлежат этому графу. Круги на этой диаграмме представляют всевозможные пересечения множеств A, $\bar{A}$ с множествами B, $\bar{B}$ (для краткости пишем XY вместо $X \cap Y$). Линии, перечеркнутые один раз, соединяют множества, ребра между которыми принадлежат разрезу $S(A)$, перечеркнутые дважды – разрезу $S(B)$. «Диагональные» ребра принадлежат обоим разрезам, а ребра внутри каждого из четырех множеств не принадлежат ни одному из двух разрезов. Видно, что сумма $S(A) \oplus S(B)$ состоит из ребер, соединяющих множества $AB \cup \bar{A}\bar{B}$ и $A\bar{B} \cup \bar{A}B = \overline{AB \cup \bar{A}\bar{B}}$, т.е. $S(A) \oplus S(B) = S(AB \cup \bar{A}\bar{B})$. □

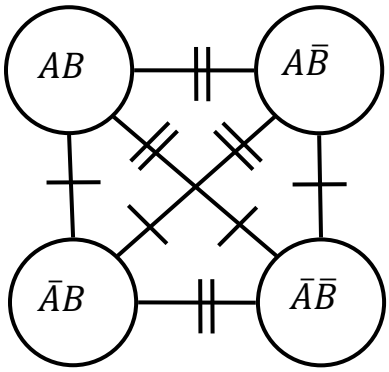


Рис. 5.5. Диаграмма Венна для доказательства теоремы о разрезах

Разрез, определяемый множеством A, состоящим из одной вершины, называется *элементарным*. Вместо $S(\{a\})$ будем писать $S(a)$.

Теорема 5.7 (о базисе пространства разрезов). *Для связного графа с n вершинами любое множество из $n-1$ элементарного разреза является базисом пространства разрезов.*

Доказательство. Рассмотрим множество $\mathcal{B}$, состоящее из всех элементарных разрезов, кроме одного. Пусть отсутствующий элементарный разрез – это $S(z)$. Заметим, что для любого множества $A \subseteq V$ выполняется равенство

$$S(A) = \sum_{x \in A} S(x)$$

(имеется в виду сумма по модулю 2). Действительно, каждое ребро, соединяющее две вершины из A, встретится в этой сумме дважды, а ребро, у которого только одна вершина принадлежит A – один раз. Из этого равенства следует, что для любого $A \subseteq V$ разрез $S(A)$ может быть выражен как сумма разрезов из $\mathcal{B}$. В самом деле, если $z \notin A$, то $S(A) = \sum_{x \in A} S(x)$, а если $z \in A$, то $S(A) = S(\bar{A}) = \sum_{x \in \bar{A}} S(x)$.

Разрезы из $\mathcal{B}$ линейно независимы, так как для любого множества $A \subseteq V - \{z\}$ разрез $S(A)$ содержит хотя бы одно ребро ввиду связности графа. Значит, никакая сумма разрезов из $\mathcal{B}$ не равна пустому графу. Значит, $\mathcal{B}$ – базис пространства разрезов. □

Для несвязного графа базис пространства разрезов можно получить, если из каждой компоненты связности взять все элементарные разрезы, кроме одного. Таким образом, размерность пространства разрезов графа с k компонентами равна $n-k$. Отметим, что размерность пространства подграфов равна сумме размерностей пространств циклов и разрезов.

Двоичные векторы $x = (x_1, x_2, \dots, x_m)$ и $y = (y_1, y_2, \dots, y_m)$ *ортогональны*, если $(x, y) = x_1 y_1 \oplus x_2 y_2 \oplus \dots \oplus x_m y_m = 0$. Два подграфа

ортогональны, если ортогональны их характеристические векторы. Ортогональность подграфов означает, что они имеют четное число общих ребер.

Теорема 5.8. *Любой квазицикл графа ортогонален любому его разрезу.*

Доказательство. Очевидно, $(x' \oplus x'', y) = (x', y) \oplus (x'', y)$. Поэтому сумма графов, каждый из которых ортогонален графу H , тоже ортогональна H. Квазицикл – это граф с четными степенями всех вершин. Степень вершины подграфа равна числу ребер, общих для этого подграфа и элементарного разреза, определяемого этой вершиной. Значит, любой квазицикл ортогонален любому элементарному разрезу. Так как любой разрез есть сумма элементарных разрезов, то любой квазицикл ортогонален любому разрезу. □

Взаимная ортогональность квазициклов и разрезов делает возможным еще один, чисто алгебраический, способ построения базиса циклов. Продемонстрируем его на примере. Пусть требуется найти базис циклов для графа, изображенного на рисунке 5.6.

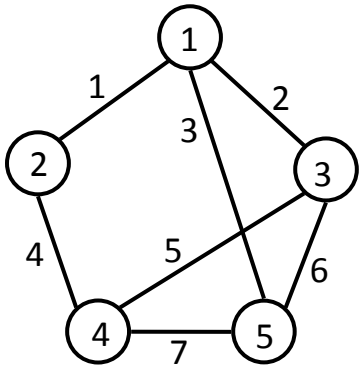


Рис. 5.6. Граф для нахождения базиса циклов

Построим матрицу инцидентности графа. Ее строки представляют собой характеристические векторы элементарных разрезов. Если удалить любую строку (удалим строку, соответствующую вершине 5), то получится матрица, строки которой представляют базис разрезов:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Так как каждый квазицикл ортогонален каждому разрезу, то остается найти фундаментальную систему решений системы линейных однородных уравнений с этой матрицей (над полем из двух элементов). Для этого найдем в полученной матрице четыре линейно независимых столбца. В общем случае для связного графа с n вершинами нужно найти $n-1$

линейно независимых столбцов. Можно доказать, что столбцы, соответствующие ребрам любого каркаса связного графа, линейно независимы, так что такое множество столбцов всегда существует. Переставим столбцы матрицы так, чтобы первыми $n-1$ столбцами были эти линейно независимые столбцы (это означает перенумерацию ребер графа). В данном примере первые четыре столбца уже линейно независимы. С помощью операций над строками добиваемся, чтобы эти четыре столбца образовывали единичную подматрицу. В данном случае можно сделать так: прибавляем первую строку ко второй, затем вторую к первой и третьей, третью ко второй, четвертую к первой и третьей. Получается матрица:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Удаляем первые четыре столбца, оставшуюся матрицу транспонируем и приписываем к ней справа единичную матрицу:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Строки полученной матрицы – характеристические векторы базисных циклов.

Задачи

5.1. Сколько существует абстрактных графов с 5 вершинами, имеющих эйлеров цикл?

5.2. Граф $K_{3,5}$ нужно превратить в граф, имеющий эйлеров цикл, изменяя (удаляя и добавляя) ребра. Каково наименьшее число изменений, если разрешается а) только удалять ребра; б) только добавлять ребра; в) и удалять, и добавлять?

5.3. Проследите работу алгоритма построения эйлерова цикла на графе $K_{4,4}$. Вершины одной доли имеют номера 1-4, другой – номера 5-8, старт в вершине 1. Всякий раз, когда имеется несколько непройденных ребер, по которым можно продолжить движение, выбирается то из них, которое ведет в вершину с наименьшим номером.

5.4. Алгоритм построения эйлерова цикла применяется к графу P_n. Каков будет ответ (содержимое стека *C* в конце работы алгоритма)?

5.5. Что нужно изменить в алгоритме построения эйлерова цикла, чтобы получился алгоритм построения эйлерова пути в графе с двумя вершинами нечетной степени?

5.6. С помощью графа де Брёйна найдите последовательность де Брёйна 1) порядка 4 для двухбуквенного алфавита; 2) порядка 2 для четырехбуквенного алфавита.

5.7. Сколько существует абстрактных графов с 5 вершинами, имеющих гамильтонов цикл?

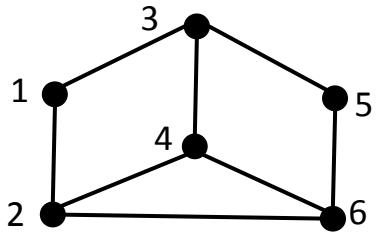
5.8. В каких из следующих графов имеется гамильтонов цикл?

1) $K_{3,3}$; 2) $K_{3,4}$; 3) $P_3 \times P_3$; 4) $P_3 \times P_4$; 5) $P_4 \times P_4$.

5.9. При каких p и q в графе $K_{p,q}$ имеется а) гамильтонов цикл? б) гамильтонов путь?

5.10. При каких n существует гамильтонов цикл в графе а) $P_3 \times P_n$; б) $P_4 \times P_n$?

5.11. Постройте дерево путей для графа, изображенного на рисунке.



5.12. Сколько листьев будет в дереве путей для графа $K_{4,4}$?

5.13. Найдите гамильтонов цикл в графе Q_4.

5.14. Найдите гамильтонов путь в турнире, заданном матрицей смежности:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

5.15. Сколько в полном графе K_n имеется подграфов, изоморфных а) C_3; б) C_4; в) C_k, $k \leq n$?

5.16. Операция сложения графов по модулю 2 может быть распространена на графы с разными множествами вершин следующим образом: если $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, то $G_1 \oplus G_2 = (V_1 \cup V_2, E_1 \otimes E_2)$.

Можно ли с помощью этой операции, а также удаления и добавления изолированных вершин и переименования вершин получить а) C_4 из C_3; б) C_5 из C_3; в) C_3 из C_4; г) C_3 из C_5?

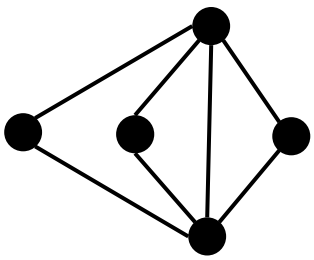
5.17. Связный граф имеет 10 вершин, из них 4 вершины степени 6, остальные – степени 3. Чему равно цикломатическое число этого графа?

5.18. Постройте фундаментальные циклы для графа $P_3 \times P_3$ относительно каркаса, полученного с помощью а) поиска в глубину; б) поиска в ширину. Выразите границу внешней грани как сумму базисных циклов.

5.19. Какова будет суммарная длина фундаментальных циклов, построенных с помощью поиска в ширину для графа а) K_n; б) $K_{n,n}$?

5.20. Сколько различных разрезов имеется а) у связного графа с n вершинами; б) у графа с n вершинами и k компонентами связности?

5.21. Найдите базис циклов по матрице инцидентности для графа, показанного на рисунке.



5.22*. Модифицируйте алгоритм построения эйлерова цикла так, чтобы не требовалась предварительная проверка четности степеней. Существование вершины нечетной степени должно обнаруживаться по ходу работы алгоритма.

5.23*. Докажите, что при любом $n \geq 2$ в графе Q_n существует гамильтонов цикл.

5.24**. Разработайте алгоритм построения дерева путей и поиска гамильтонова цикла.

5.25**. Разработайте алгоритм обхода дерева путей без его явного построения.

5.26**. Докажите, что в любом сильно связном турнире имеется гамильтонов цикл.

5.27**. Разработайте алгоритм построения базиса циклов на основе поиска в ширину.

6. Независимые множества, клики, вершинные покрытия

Независимым множеством вершин графа называется любое множество попарно не смежных вершин, т.е. множество вершин, порождающее пустой подграф. Независимое множество называется *наибольшим*, если оно содержит наибольшее количество вершин. Число вершин в наибольшем независимом множестве графа G обозначается через $\alpha(G)$ и называется *числом независимости* графа. Задача о независимом множестве состоит в нахождении наибольшего независимого множества.

Кликой графа называется множество вершин, порождающее полный подграф, т.е. множество вершин, каждые две из которых смежны. Число вершин в клике наибольшего размера называется *кликовым числом* графа и обозначается через $\omega(G)$. Очевидно, задача о независимом множестве преобразуется в задачу о клике и наоборот простым переходом от данного графа G к дополнительному графу $\bar{G}$, так что $\alpha(G) = \omega(\bar{G})$, $\omega(G) = \alpha(\bar{G})$.

Вершинное покрытие графа – это такое множество вершин, что каждое ребро графа инцидентно хотя бы одной из этих вершин. Наименьшее число вершин в вершинном покрытии графа G обозначается через $\beta(G)$и называется *числом вершинного покрытия* графа.

Все три определения иллюстрирует рисунок 6.1.

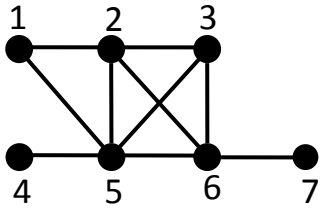


Рис. 6.1. В этом графе {1,3,4,7} – наибольшее независимое множество, {2,3,5,6} – наибольшая клика, {2,5,6} – наименьшее вершинное покрытие

Между задачами о независимом множестве и вершинном покрытии имеется простая связь.

Теорема 6.1. *Пусть G – граф с множеством вершин V. Множество $U \subseteq V$ является вершинным покрытием графа G тогда и только тогда, когда $\overline{U} = V - U$ – независимое множество.*

Доказательство. Если U – вершинное покрытие, то из двух вершин каждого ребра хотя бы одна принадлежит U. Значит, нет ребер, соединяющих две вершины из $\overline{U}$ и, следовательно, $\overline{U}$ – независимое множество. Обратно, если $\overline{U}$ – независимое множество, то нет ребер, соединяющих две вершины из $\overline{U}$. Значит, хотя бы одна из двух вершин каждого ребра принадлежит U и, следовательно, U – вершинное покрытие. □

Из этой теоремы следует, что $\alpha(G) + \beta(G) = n$ для любого графа G с n вершинами.

Итак, все три задачи тесно связаны друг с другом и алгоритм для решения одной из них легко преобразуется в алгоритм решения другой. В то же время эти задачи относятся к категории неподдающихся, так что все известные алгоритмы получения их точного решения имеют переборный характер и требуют во многих случаях много времени даже для не очень больших графов. Для того, чтобы можно было решить задачу за реальное время, приходится ослаблять требования, например, искать не точное, а достаточно хорошее или приемлемое для практики решение (например, не обязательно наибольшее, но достаточно большое независимое множество). Алгоритмы, реализующие такой подход, основываются на разнообразных интуитивных идеях о том, как найти хорошее решение. Такие идеи называют *эвристиками*, а основанные на них алгоритмы – *эвристическими алгоритмами*. Иногда удается оценить точность приближения, тогда говорят о *приближенном алгоритме*. Рассмотрим некоторые точные, эвристические и приближенные алгоритмы для задач о независимом множестве и вершинном покрытии.

Алгоритм для задачи о независимом множестве (дерево решений). Пусть G – граф, в котором требуется найти наибольшее независимое множество. Выберем в нем произвольную вершину a, не являющуюся изолированной и рассмотрим две возможности – эта вершина принадлежит искомому множеству X или не принадлежит ему. Если не принадлежит, то X является независимым множеством графа, получаемого удалением вершины a из графа G. Обозначим этот граф через G_1. Если же a принадлежит X, то никакая вершина, смежная с a, не принадлежит X. В этом случае множество X является независимым множеством графа, получающегося удалением из G всех вершин, смежных с a. Обозначим этот граф через G_2. В графе G_1 вершин меньше, чем в графе G, а так как вершина a не изолированная, то в графе G_2 их тоже меньше. Таким образом, задача о независимом множестве для графа G свелась к решению той же задачи для двух графов меньшего размера. Это приводит к рекуррентному соотношению для числа независимости

$$\alpha(G) = \max(\alpha(G_1), \alpha(G_2))$$

и к рекурсивному алгоритму для нахождения наибольшего независимого множества графа G: найдем наибольшее независимое множество X_1 графа G_1, затем наибольшее независимое множество X_2 графа G_2 и выберем большее из этих двух множеств. Процесс решения можно изобразить в виде бинарного дерева (его называют *деревом подзадач, деревом вариантов, деревом решений*). Корнем дерева служит граф G, его двумя сыновьями – графы G_1 и G_2. Чтобы не путать вершины дерева и вершины графа, вершины дерева будем называть *узлами*. Узел, не являющийся

листом, называется *внутренним узлом*. Каждому внутреннему узлу дерева соответствует некоторый граф H и некоторая вершина этого графа x. Вершину x можно выбирать произвольно, но она не должна быть изолированной вершиной графа H. Внутренний узел имеет двух сыновей – левого и правого. Левому сыну соответствует подграф графа H, получаемый удалением вершины x, а правому – подграф, получаемый удалением всех вершин, смежных с x. Листьям соответствуют подграфы, не имеющие ребер, то есть подграфы, у которых все вершины изолированные. Множества вершин этих подграфов – это независимые множества исходного графа.

Дерево решений может быть очень большим. Например, для графа mK_2 в этом дереве будет 2^m листьев. Для нахождения наибольшего независимого множества не обязательно строить все дерево полностью, а достаточно обойти его в том или ином порядке, запоминая на каждом шаге только небольшую часть информации об устройстве этого дерева. Можно, например, применить поиск в глубину для обхода дерева: сначала пройти от корня до некоторого листа, затем вернуться к предку этого листа и искать следующий лист, и т.д.

Две эвристики для задачи о независимом множестве. Одна из эвристических идей состоит в том, чтобы рассмотреть только один путь от корня до листа в дереве решений в надежде, что этому листу соответствует достаточно большое независимое множество. Для выбора этого единственного пути могут применяться разные соображения. В дереве решений, описанном выше, у каждого внутреннего узла имеются два сына. Одному из них соответствует подграф, получающийся удалением некоторой произвольно выбранной вершины a, а другому – подграф, получающийся удалением окрестности этой вершины. Чтобы вместо дерева получился один путь, достаточно каждый раз выполнять какую-нибудь одну из этих двух операций. Рассмотрим оба варианта.

Допустим, мы решили каждый раз удалять выбранную вершину. Эти удаления производятся до тех пор, пока не останется граф без ребер, т.е. независимое множество. Оно и принимается в качестве решения задачи. Для полного описания алгоритма необходимо еще сформулировать правило выбора активной вершины a. Мы хотим получить граф без ребер, в котором было бы как можно больше вершин. Чем меньше вершин будет удалено, тем больше их останется. Значит, цель – как можно быстрее удалить все ребра. Кажется, мы будем двигаться в нужном направлении, если на каждом шаге будем удалять наибольшее возможное на этом шаге число ребер. Это означает, что в качестве активной вершины всегда нужно выбирать вершину наибольшей степени.

Другой вариант – каждый раз удалять окрестность активной вершины a. Это повторяется до тех пор, пока оставшиеся вершины не будут образовывать независимого множества. Удаление окрестности вершины a равносильно тому, что сама эта вершина включается в

независимое множество, которое будет получено в качестве ответа. Так как мы хотим получить в итоге как можно большее независимое множество, следует стараться удалять на каждом шаге как можно меньше вершин. Это означает, что в качестве активной вершины всегда нужно выбирать вершину наименьшей степени.

Имеется немало графов, для которых каждая из этих эвристик дает близкое к оптимальному, а иногда и оптимальное решение. Но, как это обычно бывает с эвристическими алгоритмами, можно найти примеры графов, для которых найденные решения будут весьма далеки от оптимальных. Рассмотрим граф G_t, у которого множество вершин V состоит из трех частей: $V = A \cup B_1 \cup B_2$, причем $|A| = |B_1| = |B_2| = t$, A является независимым множеством, каждое из множеств B_1, B_2 – кликой, и каждая вершина из множества A смежна с каждой вершиной из множества $B_1 \cup B_2$. Этот граф можно представить формулой $G_t = 2K_t \circ O_t$. Степень каждой вершины из множества A в этом графе равна $2t$, а степень каждой вершины из множества $B_1 \cup B_2$ равна $2t - 1$. Первый алгоритм, выбирающий вершину наибольшей степени, будет удалять вершины из множества A до тех пор, пока не удалит их все. После этого останется граф, состоящий из двух клик, и в конечном итоге будет получено независимое множество из двух вершин. Второй алгоритм на первом шаге возьмет в качестве активной одну из вершин множества $B_1 \cup B_2$ и удалит всю ее окрестность. В результате получится граф, состоящий из этой вершины и клики, а после второго шага получится независимое множество, состоящее опять из двух вершин. Итак, при применении к этому графу любой из двух эвристик получается независимое множество из двух вершин. В то же время в графе имеется независимое множество A мощности t.

Алгоритм для задачи о вершинном покрытии (преобразования формул). Дерево решений – не единственный способ организации перебора вариантов для сложных задач. Рассмотрим алгоритм, использующий алгебраические преобразования. Для его описания удобнее рассмотреть задачу о вершинном покрытии. Пусть дан граф с множеством вершин V_n. Обозначим через $(x_1, \dots, x_n)$ характеристический вектор искомого множества X. Это множество должно быть вершинным покрытием, т.е. хотя бы одна из двух вершин каждого ребра должна принадлежать множеству X. Это означает, что для каждого ребра (i, j) дизъюнкция $x_i \vee x_j$ должна быть равна 1. Значит, должно выполняться равенство

$$\prod_{(i,j)\in E} (x_i \vee x_j) = 1,$$

где E – множество ребер графа, а П обозначает конъюнкцию. Формула в левой части равенства – конъюнктивная нормальная форма. Если раскрыть

скобки то получится дизъюнктивная нормальная форма, каждое слагаемое которой представляет некоторое вершинное покрытие графа – слагаемое $x_{i_1}x_{i_2}\dots x_{i_k}$ представляет вершинное покрытие, состоящее из вершин с номерами $i_1, i_2, \dots, i_k$. Остается выбрать слагаемое с наименьшим числом сомножителей. К сожалению, при раскрытии скобок могут получаться очень длинные формулы, даже при применении известных упрощающих преобразований вроде закона поглощения.

Приближенный алгоритм для задачи о вершинном покрытии. Рассмотрим еще один алгоритм для задачи о вершинном покрытии. Он не всегда дает точное решение, но в любом случае получаемое решение не более чем в два раза отличается от оптимального

Работа алгоритма начинается с создания пустого множества X и состоит в выполнении однотипных шагов, в результате каждого из которых к множеству X добавляются новые вершины. Допустим, перед очередным шагом имеется некоторое множество вершин X. Если оно покрывает все ребра (т.е. каждое ребро инцидентно одной из этих вершин), то процесс заканчивается и множество X принимается в качестве искомого вершинного покрытия. В противном случае выбирается какое-нибудь непокрытое ребро (a, b), и вершины a и b добавляются к множеству X.

Для полного описания алгоритма нужно еще сформулировать правило выбора ребра (a, b). Однако для оценки степени приближения, которая будет сейчас получена, это не имеет значения. Можно считать, что какое-то правило принято.

Обозначим через $\beta'(G)$ мощность вершинного покрытия, которое получится при применении этого алгоритма к графу G, и докажем, что $\beta'(G) \le 2\beta(G)$. Действительно, допустим, что до окончания работы алгоритм выполняет k шагов, добавляя к множеству X пары вершин $(a_1, b_1), \dots, (a_k, b_k)$. Тогда $\beta'(G) = 2k$. Никакие два из этих k ребер не имеют общей вершины. Значит, чтобы покрыть все эти ребра, нужно не меньше k вершин. Следовательно, $\beta(G) \ge k$ и $\beta'(G) \le 2\beta(G)$.

Задачи

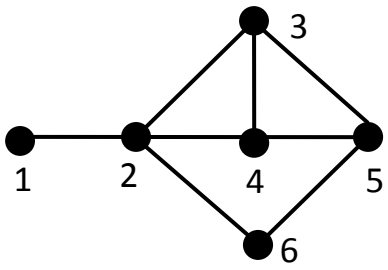
6.1. Сколько имеется абстрактных графов с $\alpha(G) = 3$, имеющих гамильтонов цикл а) с 5 вершинами; б) с $\omega(G) = 2$ и 6 вершинами?

6.2. Найдите 1) $\alpha((C_7 \circ P_7) + Q_3)$; 2) $\alpha(C_3 \times C_5)$; 3) $\beta(P_3 \times P_4)$.

6.3. Найдите $\alpha(Q_n)$, $\beta(Q_n)$, $\omega(Q_n)$.

6.4. Сколько наибольших независимых множеств имеется у графа а) C_8, б) C_9; в) P_8; г) P_9?

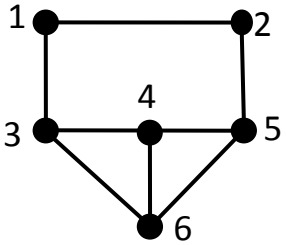
6.5. Найдите наибольшее независимое множество с помощью дерева решений в графе, показанном на рисунке.



6.6. Сколько листьев будет в дереве решений для независимых множеств, построенном для графа pK_3?

6.7. Рассмотрим две эвристики для задачи о независимом множестве:
1) выбирать вершину наибольшей степени и удалять ее, пока не останется независимое множество;
2) выбирать вершину наименьшей степени и удалять ее окрестность, пока не останется независимое множество.
Какая из них всегда дает правильный ответ а) для графа $K_{p,q}$; б) для графа P_n; в) для графа $(K_1 + K_2) \circ O_3$?

6.8. Найдите наименьшее вершинное покрытие алгебраическим методом в графе, показанном на рисунке. Примените к нему также приближенный алгоритм.



6.9. Даны графы G_1, G_2 и G_3, с 9 вершинами каждый. Известно, что $\alpha(G_1) = 2,\ \alpha(G_2) = 3, \alpha(G_3) = 4$. Найдите
1) $\alpha(G_1 + G_2 + G_3)$;
2) $\alpha(G_1 \circ G_2 \circ G_3)$;
3) $\alpha((G_1 + G_2) \circ G_3)$;
4) $\beta(G_1 + G_2 + G_3)$;
5) $\beta((G_1 \circ G_2) + G_3)$.

6.10. Какие из следующих равенств и неравенств выполняются для любых графов G_1 и G_2:
1) $\alpha(G_1 \times G_2) = \alpha(G_1)\alpha(G_2)$;
2) $\alpha(G_1 \times G_2) \geq \alpha(G_1)\alpha(G_2)$;
3) $\alpha(G_1 \times G_2) = \max(\alpha(G_1), \alpha(G_2))$;
4) $\omega(G_1 \times G_2) = \omega(G_1)\omega(G_2)$

5) $\omega(G_1 \times G_2) \geq \omega(G_1)\omega(G_2)$;
6) $\omega(G_1 \times G_2) = \max(\omega(G_1), \omega(G_2))$?

6.11. Существует ли граф, у которого $\alpha(G) = 2$ и $\omega(G) = 2$ а) с 5 вершинами; б) с 6 вершинами?

6.12*. Докажите, что для любого графа G с 10 вершинами выполняется хотя бы одно из неравенств $\alpha(G) \geq 4$ или $\omega(G) \geq 3$.

6.13*. Докажите, что вторая эвристика из задачи 6.8, если ее применить к дереву, всегда дает наибольшее независимое множество. Верно ли это для первой эвристики из той же задачи?

6.14*. На прямой задано конечное множество интервалов, требуется найти наибольшее подмножество попарно не пересекающихся интервалов. Покажите, что задача сводится к нахождению наибольшего независимого множества в графе. Разработайте алгоритм, решающий эту задачу за время $O(n)$, где n – число интервалов.

7. Паросочетания

7.1. Паросочетания и реберные покрытия

Паросочетанием в графе называется множество ребер, попарно не имеющих общих вершин. Задача о паросочетании состоит в том, чтобы в данном графе найти паросочетание с наибольшим числом ребер. Это число для графа G будем обозначать через $\pi(G)$.

Реберным покрытием графа называется такое множество ребер, что всякая вершина графа инцидентна хотя бы одному из этих ребер. Наименьшее число ребер в реберном покрытии графа G обозначим через $\rho(G)$. Заметим, что реберное покрытие существует только у графов, не имеющих изолированных вершин.

Определение паросочетания похоже на определение независимого множества вершин, паросочетание иногда так и называют – независимое множество ребер. Эта аналогия еще усиливается тесной связью между реберными покрытиями и паросочетаниями, подобно тому, как связаны между собой вершинные покрытия и независимые множества. Даже равенство, количественно выражающее эту связь, имеет точно такой же вид (напомним, что числа независимости $\alpha(G)$ и вершинного покрытия $\beta(G)$ связаны равенством $\alpha(G) + \beta(G) = n$).

Теорема 7.1. *Для любого графа G с n вершинами, не имеющего изолированных вершин, справедливо равенство $\pi(G) + \rho(G) = n$.*

Доказательство. Пусть M – наибольшее паросочетание в графе G, т.е. $|M| = \pi(G)$. Обозначим через W множество всех вершин графа, не покрытых ребрами паросочетания M. Тогда $|W| = n - 2\pi(G)$. Очевидно, W – независимое множество (иначе M не было бы наибольшим). Выберем для каждой вершины из W какое-нибудь инцидентное ей ребро. Пусть N – множество всех выбранных ребер. Тогда $M \cup N$ – реберное покрытие и $|M \cup N| = n - \pi(G)$. Следовательно, $\rho(G) \leq n - \pi(G)$.

Докажем обратное неравенство. Пусть C – наименьшее реберное покрытие графа G. Рассмотрим подграф H графа G, образованный ребрами этого покрытия. В графе H из концевых вершин каждого ребра хотя бы одна имеет степень 1. Действительно, если у некоторого ребра в H степень обеих вершин больше 1, то множество ребер, получаемое удалением этого ребра из C, тоже будет реберным покрытием., а это противоречит минимальности покрытия C. Отсюда следует, что каждая компонента связности графа H является звездой (напомним: звезда – это полный двудольный граф, у которого одна доля состоит из одной вершины). В лесе с n вершинами и k компонентами связности число ребер равно $n - k$. H – лес и в нем $\rho(G)$ ребер, значит, у него $n - \rho(G)$ компонент связности.

Выбрав по одному ребру из каждой компоненты, получим паросочетание. Отсюда следует, что $\pi(G) \geq n - \rho(G)$. □

Из этого доказательства видно, как получить решение одной из этих двух задач, если известно решение другой. Если известно наибольшее паросочетание M, то нужно для каждой вершины, не покрытой ребрами паросочетания, взять какое-нибудь ребро, инцидентное этой вершине и добавить к множеству M. Полученное множество ребер будет наименьшим реберным покрытием. Обратно, если C – наименьшее реберное покрытие, нужно в подграфе, образованном ребрами этого покрытия, найти компоненты связности и выбрать по одному ребру из каждой компоненты. Получится наибольшее паросочетание.

Несмотря на такое сходство между "вершинными" и "реберными" вариантами независимых множеств и покрытий, имеется кардинальное различие в сложности соответствующих экстремальных задач. "Вершинные" задачи относятся к разряду неподдающихся, для реберных же известны эффективные алгоритмы.

7.2. Метод чередующихся путей

Пусть G – граф, M – паросочетание в нем. Ребра паросочетания будем называть *сильными,* остальные ребра графа – *слабыми*. Вершину назовем *свободной,* если она не принадлежит ребру паросочетания. На рисунке 7.1 слева показан граф и в нем выделены ребра паросочетания $M = \{(2,3), (5,6), (7,8)\}$. Вершины 1 и 4 – свободные. Заметим, что к этому паросочетанию нельзя добавить ни одного ребра (такое паросочетание называют максимальным). Однако оно не является наибольшим. В этом легко убедиться, если рассмотреть путь 1,5,6,7,8,4 (показан пунктиром). Он начинается и заканчивается в свободных вершинах, а вдоль пути чередуются сильные и слабые ребра. Если на этом пути превратить каждое сильное ребро в слабое, а каждое слабое – в сильное, то получится новое паросочетание, показанное на рисунке справа, в котором на одно ребро больше. Увеличение паросочетания с помощью подобных преобразований – в этом и состоит суть метода чередующихся путей.

Рис. 7.1. Увеличивающий путь и увеличенное паросочетание

Сформулируем необходимые понятия и докажем теорему, лежащую в основе этого метода. *Чередующимся путем* относительно данного паросочетания называется простой путь, в котором чередуются сильные и слабые ребра (т.е. за сильным ребром следует слабое, за слабым – сильное). Чередующийся путь называется *увеличивающим*, если он соединяет две свободные вершины. Пусть M – паросочетание, P – чередующийся путь относительно M. Операцию превращения всех слабых ребер пути P в сильные, а сильных – в слабые, будем называть *инверсией* вдоль пути P.

Теорема 7.2 (об увеличивающем пути). *Паросочетание является наибольшим тогда и только тогда, когда относительно него нет увеличивающих путей.*

Доказательство. Пусть M – паросочетание, P – увеличивающий путь относительно M. Выполним инверсию вдоль пути P. Пусть M' – множество всех сильных ребер после инверсии. Концевые вершины пути P до инверсии были свободными, после нее каждая из них становится инцидентной одному сильному ребру. Каждая из остальных вершин пути как была, так и останется инцидентной одному сильному ребру. Значит, M' – паросочетание. До инверсии оба концевых ребра пути P были слабыми, значит, слабых ребер на этом пути было на одно больше, чем сильных. После инверсии сильных станет на одно больше. Следовательно, $|M'| = |M| + 1$ и M – не наибольшее паросочетание.

Докажем обратное утверждение. Пусть M – паросочетание в графе G, и M не наибольшее. Покажем, что тогда имеется увеличивающий путь относительно M. Существует паросочетание M' такое, что $|M'| > |M|$. Рассмотрим подграф H графа G, образованный теми ребрами, которые входят в одно и только в одно из паросочетаний M, M'. Иначе говоря, множеством ребер графа H является симметрическая разность $M \otimes M'$. В графе H каждая вершина инцидентна не более чем двум ребрам (одному из M и одному из M'), т.е. имеет степень не более двух. В таком графе каждая компонента связности – путь или цикл. В каждом из этих путей и циклов чередуются ребра из M и M'. Так как $|M'| > |M|$, то имеется компонента, в которой ребер из M' содержится больше, чем ребер из M. Это может быть только путь, у которого оба концевых ребра принадлежат M'. Но тогда относительно паросочетания M этот путь будет увеличивающим. □

Для решения задачи о паросочетании остается научиться находить увеличивающие пути или убеждаться, что таких путей нет. Это можно сделать, перебирая чередующиеся пути (подобно перебору путей при поиске гамильтонова цикла). Дело осложняется тем, что чередующихся путей может быть много (см. задачу 7.7). Оказывается, нет необходимости перебирать их все. Известны эффективные алгоритмы, которые ищут

увеличивающие пути для произвольных графов. Рассмотрим простой алгоритм, решающий эту задачу для двудольных графов.

7.3. Паросочетания в двудольных графах

Известны разнообразные интерпретации задачи о наибольшем паросочетании в двудольном графе в образах реального мира. Одна из них – задача о назначении на должности. Имеется некоторое количество вакантных должностей и множество кандидатов на эти должности. Должности и люди образуют две доли двудольного графа, а ребро соединяет человека с должностью, если данный человек по своей квалификации способен выполнять данную работу. Наибольшее паросочетание позволяет заполнить максимальное количество вакансий и трудоустроить максимальное число соискателей.

Пусть G – двудольный граф с долями A и B, M – паросочетание в G. Всякий увеличивающий путь, если такой имеется, соединяет вершину из множества A с вершиной из B. Поэтому при поиске увеличивающего пути достаточно рассматривать чередующиеся пути, начинающиеся в свободных вершинах из доли A. Если в одной из долей свободных вершин нет, то увеличивающих путей не существует и данное паросочетание – наибольшее.

Зафиксируем некоторую свободную вершину $a \in A$ и будем искать увеличивающий путь, начинающийся в этой вершине. Заметим, что в чередующемся пути, ведущем из вершины a в вершину из доли B, последнее ребро обязательно слабое, а в вершину из доли A – сильное.

Дерево T с корнем в вершине a назовем *деревом достижимости* для вершины a, если

1) T является подграфом графа G;
2) каждый путь в дереве T, начинающийся в корне, является чередующимся путем;
3) T – максимальное дерево со свойствами 1) и 2).

Дерево достижимости определено не однозначно, но любое такое дерево в двудольном графе обладает следующим свойством.

Утверждение 1. *Вершина x принадлежит дереву достижимости для вершины a тогда и только тогда, когда существует чередующийся путь, соединяющий вершины a и x.*

Доказательство. Рассмотрим некоторое дерево достижимости T с корнем a и докажем, что всякая вершина x, достижимая из вершины a чередующимся путем, принадлежит этому дереву. Проведем индукцию по длине кратчайшего чередующегося пути из a в x. Пусть P – такой путь, y – предпоследняя (т.е. предшествующая x) вершина пути P. По

предположению индукции вершина y принадлежит дереву T. Пусть z – отец вершины y в дереве T. Допустим, ребро (y, x) слабое. Тогда вершина x принадлежит доле B, а вершина y – доле A. Следовательно, любой чередующийся путь из вершины a в вершину y заканчивается сильным ребром. Это относится и к пути между вершинами a и y в дереве T. Значит, ребро (z, y) сильное. Поэтому, если добавить к дереву вершину x и ребро (y, x), то путь, соединяющий вершины a и x в дереве, будет чередующимся. Значит, если предположить, что вершина x не принадлежит дереву, то окажется, что дерево T не максимально, а это противоречит определению дерева достижимости. Аналогично рассматривается случай, когда ребро (y, x) сильное. □

Для построения дерева достижимости можно использовать слегка модифицированный поиск в ширину из вершины a. Отличие от стандартного поиска в ширину состоит в том, что для вершин из доли A (они находятся на четном расстоянии от вершины a) исследуются не все инцидентные им ребра, а только слабые, а для вершин из доли B – только сильные. Впрочем, с таким же успехом можно применить поиск в глубину. Деревья могут получиться разными, но у них будет одно и то же множество вершин. Пример показан на рисунке 7.2.

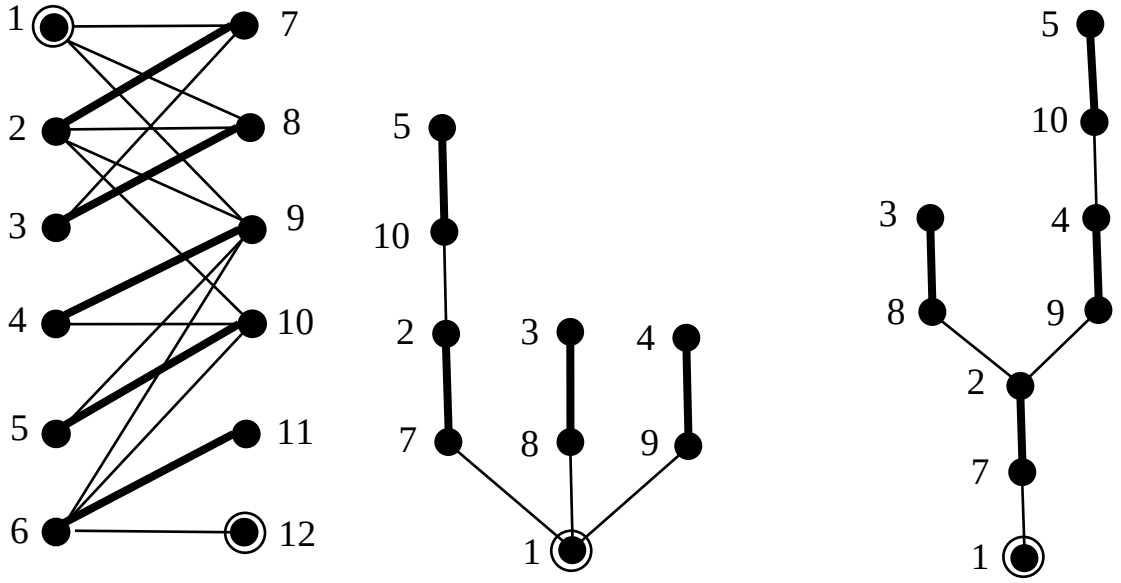


Рис. 7.2. Двудольный граф с паросочетанием в нем и два дерева достижимости с корнем в вершине 1. Левое построено поиском в ширину, правое – поиском в глубину. Увеличивающего пути нет, значит, данное паросочетание наибольшее

Если в дереве достижимости появляется отличная от a свободная вершина b, это означает, что найден увеличивающий путь – это путь между вершинами a и b в дереве. Выполнив инверсию, получим большее паросочетание. После этого, если в обеих долях еще имеются свободные вершины, возобновляется поиск увеличивающего пути.

Если дерево достижимости построено и в нем нет других свободных вершин, кроме корня, то нужно выбрать другую свободную вершину в доле A (если такие еще есть) и строить дерево достижимости для нее. При этом, как показывает следующее утверждение, вершины первого дерева можно не рассматривать (временно пометить как удаленные).

Утверждение 2. *Если дерево достижимости содержит хотя бы одну вершину увеличивающего пути, то оно содержит увеличивающий путь.*

Доказательство. Допустим, дерево достижимости T с корнем a имеет общие вершины с увеличивающим путем P, соединяющим свободные вершины b и c. Покажем, что в T есть увеличивающий путь. Достаточно доказать, что хотя бы одна из вершин b, c принадлежит дереву T. Если одна из них совпадает с вершиной a, то согласно утверждению 1 другая тоже принадлежит дереву, так что в этом случае в дереве имеется увеличивающий путь между вершинами b и c. Допустим, обе вершины b и c отличны от a. Пусть R – путь в дереве T, начинающийся в вершине a и заканчивающийся в вершине x, принадлежащей пути P, и не содержащий других вершин пути P. Очевидно, последнее ребро пути R слабое. Вершина x делит путь P на два отрезка, P_b и P_c, содержащие соответственно вершины b и c. В одном из этих отрезков, скажем, в P_b, ребро, инцидентное вершине x, сильное. Тогда объединение путей R и P_b образует чередующийся путь, соединяющий вершины a и b. Значит, вершина и принадлежит дереву T. □

В качестве одной итерации алгоритма можно рассматривать поиск одного увеличивающего пути (при этом может быть построено несколько деревьев достижимости) и выполнение инверсии при положительном исходе поиска. Каждая итерация, кроме последней, увеличивает мощность паросочетания на единицу. Согласно утверждению 2 при построении очередного дерева достижимости вершины всех деревьев достижимости, построенных ранее в рамках той же итерации, можно не рассматривать. Поэтому при поиске в ширину или в глубину каждое ребро в ходе одной итерации будет исследовано не более одного раза и общая трудоемкость одной итерации оценивается как $O(m)$. Число ребер в паросочетании не превышает $n/2$, значит, и число итераций не превосходит этой величины. Таким образом, трудоемкость алгоритма в целом можно оценить как $O(mn)$.

7.4. Независимые множества в двудольных графах

Метод увеличивающих путей может применяться и для решения других задач. Рассмотрим задачу о независимом множестве. Она трудна в общем случае, но для двудольных графов допускает эффективное решение.

Пусть G – двудольный граф с долями A и B и требуется найти в нем наибольшее независимое множество. Найдем сначала наибольшее паросочетание M в этом графе. Пусть A_0 – множество свободных вершин в доле A. Найдем все вершины из A, достижимые чередующимися путями из A_0, пусть A_1 – множество всех таких вершин а B_1 – множество других концов ребер паросочетания, инцидентных вершинам из A_1. Обозначим $A_2 = A - (A_0 \cup A_1)$, $B_2 = B - B_1$ (см. рисунок 7.3).

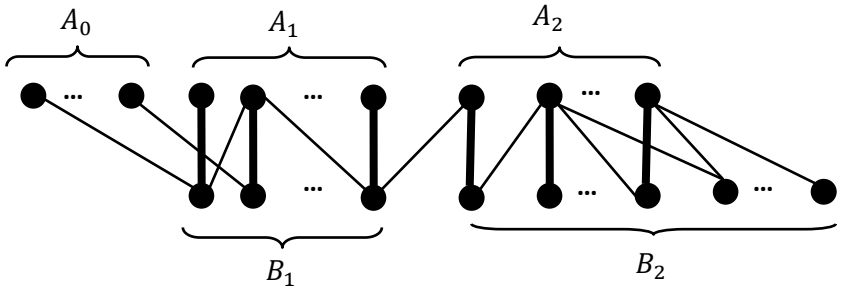


Рис. 7.3. Разбиение множества вершин при заданном паросочетании

Вершины из B_2 не смежны с вершинами из $A_0 \cup A_1$, так как иначе они были бы достижимы из A_0. Значит, $A_0 \cup A_1 \cup B_2$– независимое множество. Оно содержит все свободные вершины и по одной вершине из каждого ребра паросочетания. Отсюда следует, что это независимое множество – наибольшее. Дополнительное к нему множество $A_2 \cup B_1$ является наименьшим вершинным покрытием. Оно состоит в точности из $\pi(G)$ вершин, поэтому справедливо следующее утверждение.

Теорема 7.3 (теорема Кёнига-Эгервари). *Для любого двудольного графа G выполняется равенство $\beta(G) = \pi(G)$.*

Эта теорема представляет собой графовый вариант знаменитой теоремы Кёнига-Эгервари о матрицах. Для задания двудольного графа вместо всей матрицы смежности достаточно взять ее подматрицу, в которой строки соответствуют вершинам одной доли, а столбцы – вершинам другой. Назовем ее *сокращенной матрицей смежности* двудольного графа. Всякая прямоугольная (0,1)-матрица (т.е. матрица, составленная из элементов 0,1) является сокращенной матрицей смежности некоторого двудольного графа. Строки и столбцы матрицы будем называть *линиями*. Линии сокращенной матрицы смежности соответствуют вершинам графа, а единичные элементы матрицы – его ребрам, Если взять вершинное покрытие двудольного графа, то в

сокращенной матрице смежности ему соответствует множество линий, содержащих все единицы этой матрицы. Паросочетанию графа соответствует множество единичных элементов матрицы, никакие два из которых не находятся на одной линии. Поэтому, если перевести теорему 7.3 на язык матриц, получается следующее утверждение.

Теорема 7.4 (матричная форма теоремы Кёнига-Эгервари). *В любой (0,1)-матрице наименьшее число линий, содержащих все единицы, равно наибольшему числу единиц, никакие две из которых не находятся на одной линии.*

7.3. Паросочетания в произвольных графах

Для графа, не являющегося двудольным, утверждение 1 может быть неверным. Пример этого показан на рисунке 7.4. В графе, изображенном слева, имеется увеличивающий путь 5, 3, 1, 2, 4, 6. Справа показано дерево достижимости, построенное для вершины 5. Вершина 6 не вошла в это дерево, хотя имеется чередующийся путь, соединяющий ее с вершиной 5. В результате увеличивающий путь не будет найден. Причина этого – наличие в графе ребра (1, 2). В тот момент, когда исследуется это ребро, обе вершины 1 и 2 уже присутствуют в дереве, поэтому построение дерева заканчивается. Ребро (1, 2) замыкает нечетный цикл, и существование такого цикла является настоящей причиной неудачи. Таким образом, может случиться, что существует чередующийся путь, соединяющий вершину x с вершиной a, но x не войдет в дерево достижимости для a.

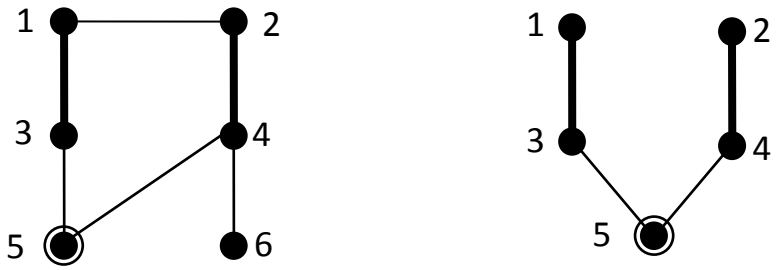


Рис. 7.4. Утверждение 1 не всегда верно для произвольных графов

Тем не менее, и для графов с нечетными циклами задачу о наибольшем паросочетании можно решать эффективно. Первый алгоритм нахождения наибольшего паросочетания в произвольном графе за полиномиальное время был предложен Эдмондсом в 1965 г. Опишем этот алгоритм, не приводя его обоснования.

Алгоритм Эдмондса. Пусть имеется граф G и в нем паросочетание M. Сначала, как и в случае двудольного графа, методом поиска в ширину

строим дерево достижимости для некоторой свободной вершины a. При этом, как и в двудольном случае, для вершин, находящихся в дереве на четном расстоянии от корня, исследуем только слабые ребра, а для вершин с нечетным расстоянием от корня – только сильные. Построение дерева для двудольного графа прекращалось, если к дереву нельзя было добавить ни одной вершины, либо если к нему добавлялась свободная вершина, т.е. обнаруживался увеличивающий путь. Для произвольного графа будем прекращать построение дерева еще и в том случае, когда исследуемое на очередном шаге ребро соединяет две вершины дерева, находящиеся на одинаковом расстоянии от корня. Обнаружение такого ребра означает, что найден подграф, называемый *цветком* (см. рис 7.5). Он состоит из чередующегося пути P, соединяющего корень дерева a с некоторой вершиной b, и нечетного цикла C. При этом b является единственной общей вершиной пути P и цикла C, а C можно рассматривать как замкнутый чередующийся путь, начинающийся и заканчивающийся в вершине b.

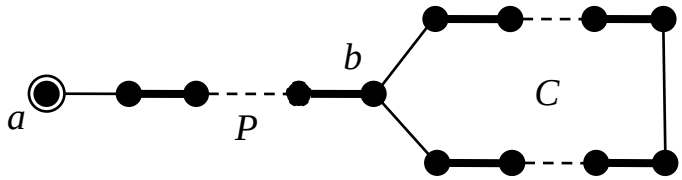


Рис. 7.5. Цветок в графе

Можно доказать, что если существует вершина, достижимая из вершины a чередующимся путем, но не вошедшая в дерево достижимости, то в графе имеется ребро, при добавлении которого к дереву достижимости образуется цветок. Выявление цветков не представляет трудности – нужно только контролировать обратные ребра, появляющиеся при построении дерева достижимости. Если концы такого ребра находятся на одинаковом расстоянии от корня, это и означает, что найден цветок. Обнаружив цветок, выполним инверсию вдоль пути P. После этого преобразования новое множество сильных ребер останется паросочетанием той же мощности, но вместо вершины a свободной станет вершина b. Таким образом, на цикле C будет одна свободная вершина. Оказывается, такой цикл можно стянуть в одну вершину, не теряя информации о существовании увеличивающих путей. Стягивание цикла состоит в том, что стягивается каждое ребро цикла. Получаемый в результате граф обозначим через G'. В нем цикл C заменен одной вершиной, которую обозначим через b'. Ребра паросочетания M, не принадлежащие циклу C, образуют паросочетание M' в графе G'. Отметим, что b' является свободной вершиной относительно паросочетания M'. Можно доказать, что M является наибольшим паросочетанием в графе G тогда и только тогда, когда M' – наибольшее паросочетание в графе G'. Таким образом,

если окажется, что в G' не существует увеличивающего пути относительно M', то M – наибольшее паросочетание. Если же в G' будет найден увеличивающий путь, то увеличивающий путь в G можно построить следующим образом.

Пусть P' – увеличивающий путь в графе G'. Если P' не проходит через вершину b', то он будет увеличивающим путем и в графе G. В противном случае рассмотрим путь P'', получающийся удалением вершины b' из пути P'. Можно считать, что вершина b' была последней вершиной пути P'. Пусть x – предшествующая ей вершина этого пути, тогда x – последняя вершина пути P''. Так как вершина x смежна с вершиной b' в графе G', то в графе G на цикле C имеется вершина y, смежная с x. Добавим к пути P'' тот из отрезков цикла C, соединяющих вершину y с вершиной b, который начинается сильным ребром. В результате получится увеличивающий путь в графе G.

Задачи

7.1. Найдите наибольшее паросочетание и наименьшее реберное покрытие в графах P_6, P_7, C_7, $K_{1,5}$, $K_{3,7}$.

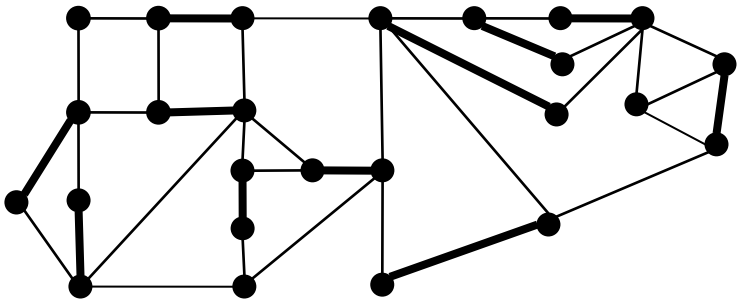
7.2. Найдите число $\rho(G)$ для графа 1) $K_1 \circ 5C_3$; 2) $K_2 \circ 4P_3$; 3) $K_3 \circ 2C_5$; 4) $C_5 \circ 3C_5$.

7.3. Найдите $\pi(K_{1,3} \times C_3)$.

7.4. Сколько наибольших паросочетаний имеется в графе 1) P_5; 2) P_6; 3) C_5; 4) C_6; 5) K_4; 6) K_5 7) $K_{3.6}$?

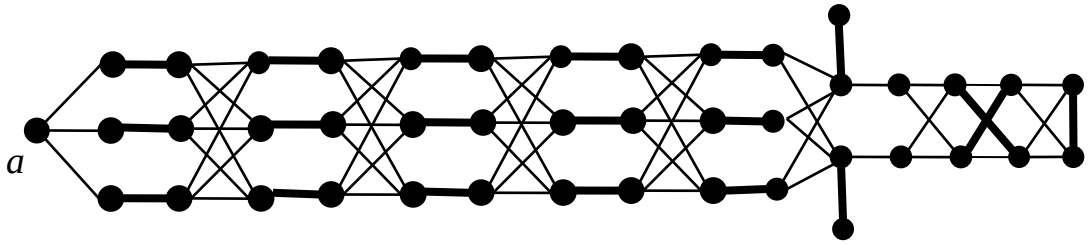
7.5. Докажите, что для любой вершины степени 1 существует наибольшее паросочетание, содержащее ребро, инцидентное этой вершине.

7.6. Является ли показанное на рисунке паросочетание наибольшим?

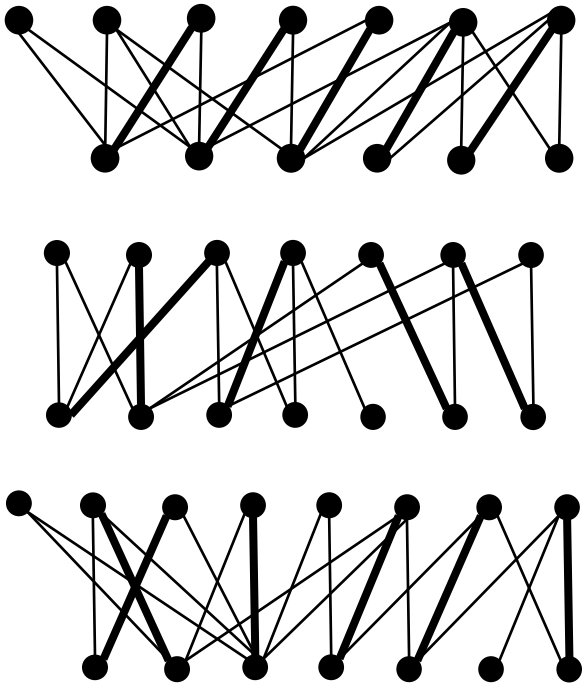


7.7. Сколько имеется максимальных (не имеющих продолжения) чередующихся путей относительно паросочетания, показанного на рисунке, начинающихся в вершине a? Есть ли среди них

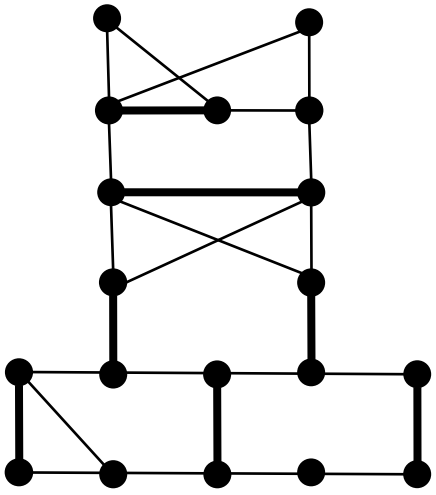
увеличивающий путь? Есть ли вообще для данного паросочетания увеличивающий путь?



7.8. Проверьте, являются ли данные паросочетания наибольшими. Найдите в этих графах наименьшие реберные покрытия и наибольшие независимые множества.



7.9. Примените алгоритм Эдмондса к графу, показанному на рисунке.



7.10*. Реализуйте алгоритм нахождения наибольшего паросочетания в двудольном графе.

7.11*. Используя решение задачи 7.5, разработайте алгоритм, который находит наибольшее паросочетание в любом дереве за время $O(n)$.

7.12**. Реализуйте алгоритм Эдмондса.

8. Раскраски

8.1. Раскраска вершин

Раскраской вершин графа называется назначение цветов его вершинам. Иначе говоря, раскраска – это функция $f: V \to C$, где V – множество вершин, C – множество цветов. Раскраска называется *правильной*, если любые две смежные вершины имеют разные цвета. Задача о раскраске состоит в нахождении правильной раскраски данного графа G в наименьшее число цветов. Это число называется *хроматическим числом* графа и обозначается $\chi(G)$.

В правильной раскраске полного графа K_n все вершины должны иметь разные цвета, поэтому $\chi(K_n) = n$. Если в каком-нибудь графе имеется полный подграф с k вершинами, то для раскраски этого подграфа необходимо k цветов. Отсюда следует, что для любого графа выполняется неравенство

$$\chi(G) \geq \omega(G).$$

Хроматическое число может быть и строго больше кликового числа. Например, $\omega(\mathrm{C}_5) = 2$, а $\chi(C_5) = 3$.

В графе с раскрашенными вершинами множество вершин одного цвета называют *цветным классом*. При правильной раскраске каждый цветной класс является независимым множеством. Поэтому правильную раскраску можно также рассматривать как разбиение множества вершин на независимые множества, а хроматическое число – как наименьшее число частей в таком разбиении. Каждый цветной класс содержит не более $\alpha(G)$ вершин. Поэтому произведение числа классов (т.е. цветов) на $\alpha(G)$ не меньше числа всех вершин графа. Отсюда следует неравенство

$$\chi(G) \geq \frac{n}{\alpha(G)}.$$

Тот же граф C_5 служит примером, когда это неравенство строгое.

Очевидно, $\chi(G) = 1$ тогда и только тогда, когда G – пустой граф. Нетрудно охарактеризовать и графы с хроматическим числом 2 (точнее, не больше 2). По определению, это такие графы, у которых множество вершин можно разбить на два независимых множества. Но это совпадает с определением двудольного графа. Поэтому двудольные графы называют еще *бихроматическими*. По теореме Кёнига (теорема 3.8) граф является бихроматическим тогда и только тогда, когда в нем нет циклов нечетной длины.

Для графов с хроматическим числом 3 такого простого описания не найдено. Неизвестны и простые алгоритмы, проверяющие, можно ли данный граф раскрасить в 3 цвета. Задача такой проверки (вообще, задача

проверки возможности раскрасить граф в k цветов при любом фиксированном $k \geq 3$) относится к неподдающимся задачам.

Алгоритм решения задачи о раскраске (дерево решений). Рассмотрим алгоритм решения задачи о раскраске, похожий на описанный выше алгоритм для задачи о независимом множестве. Сходство заключается в том, что задача для данного графа сводится к той же задаче для двух других графов. Поэтому снова возникает дерево решений, обход которого позволяет найти решение. Но есть и одно существенное различие, состоящее в том, что теперь два новых графа не будут подграфами исходного графа.

Выберем в данном графе G две несмежные вершины x и y и построим два новых графа: G_1, получающийся добавлением ребра (x, y) к графу G, и G_2, получающийся из графа G слиянием вершин x и y. Операция слияния состоит в удалении вершин x и y и добавлении новой вершины и ребер, соединяющих ее с каждой вершиной, с которой была смежна хотя бы одна из вершин x, y. Новой вершине ставится в соответствие множество $\{x, y\}$. Пример показан на рисунке 7.5.

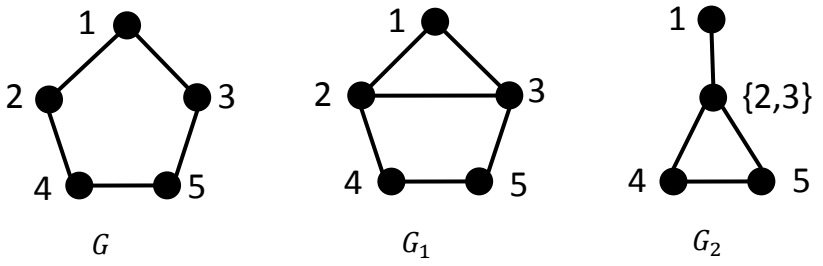


Рис. 7.5. Добавление ребра (2,3) и слияние вершин 2 и 3

Если в правильной раскраске графа G вершины x и y имеют разные цвета, то она будет правильной и для графа G_1. Если же цвета вершин x и y в раскраске графа G одинаковы, то граф G_2 можно раскрасить в то же число цветов: новая вершина $\{x, y\}$ окрашивается в тот цвет, в который окрашены вершины x и y, а все остальные вершины сохраняют те цвета, которые они имели в графе G. И наоборот, раскраска каждого из графов G_1, G_2, очевидно, дает раскраску графа G в то же число цветов. Поэтому

$$\chi(G) = \min\{\chi(G_1), \chi(G_2)\}.$$

Таким образом, задача о раскраске данного графа G сводится к той же задаче для двух графов. В одном из них на одну вершину меньше, чем в исходном графе, в другом вершин столько же, но на одно ребро больше. В каждом из этих графов снова можно выбрать пару несмежных вершин (если такие есть) и применить те же операции, получив новые графы и т.д. В результате возникает дерево решений. Каждому внутреннему узлу этого дерева соответствует некоторый граф H и пара несмежных вершин x, y

этого графа. Вершинам графа H сопоставляются подмножества множества V вершин графа G. Эти подмножества образуют разбиение множества V (для самого графа G все эти подмножества одноэлементные). Каждый внутренний узел имеет левого и правого сыновей. Выбор левого сына означает, что далее рассматриваются раскраски, в которых вершины x и y окрашиваются в разные цвета, это достигается добавлением ребра (x, y) к графу H. Правый сын соответствует раскраскам, в которых вершины x и y окрашиваются одинаково. Эти вершины сливаются в одну и этой новой вершине сопоставляется объединение множеств, сопоставленных в графе H вершинам x и y. Листьям дерева соответствуют графы, у которых нет ни одной пары несмежных вершин, т.е. полные графы. Этому графу соответствует раскраска исходного графа, в которой цветными классами служат подмножества множества V, сопоставленные вершинам графа-листа.

Последовательная раскраска. Одна из простейших эвристик для задачи о раскраске называется последовательной раскраской и состоит в следующем. Вершины графа как-нибудь линейно упорядочиваются и раскрашиваются в этом порядке. Цветами являются натуральные числа. Очередная вершина красится в наименьший цвет, который еще не использован ни для одной из смежных с ней вершин. Число цветов в получаемой таким образом раскраске зависит от того, в каком порядке раскрашиваются вершины. На рисунке 7.6. показаны результаты последовательной раскраски одного и того же графа при разных упорядочениях вершин. Числа внутри кругов, изображающих вершины, – это номера вершин в том порядке, в котором они раскрашиваются, а рядом с кругами – цвета вершин.

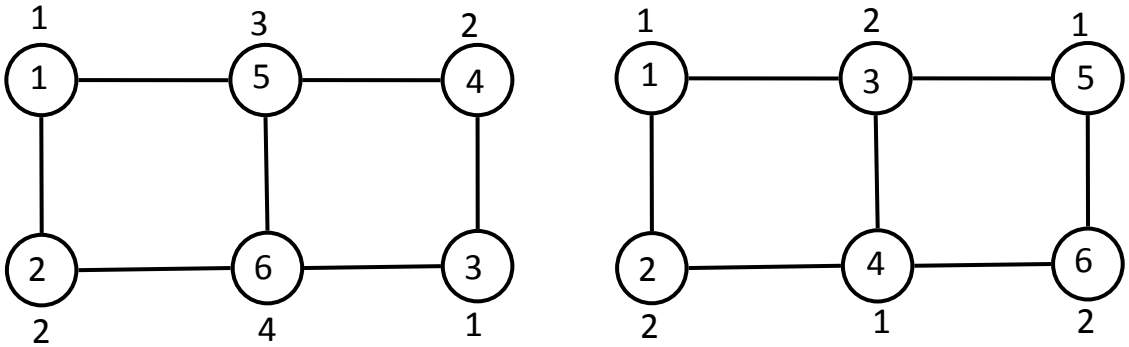


Рис.7.6. Две последовательных раскраски одного графа

Интересно, что для любого графа существует упорядочение, при котором последовательная раскраска дает оптимальный результат. Действительно, пусть $V_1, V_2, \ldots, V_k$ – цветные классы при раскраске графа в наименьшее число цветов. Если при последовательной раскраске сначала будут окрашиваться все вершины из множества V_1, затем – все из множества V_2 и т.д., то в результате получится именно эта раскраска.

Обозначим через $\Delta(G)$ наибольшую степень вершины в графе G. При окрашивании очередной вершины в алгоритме последовательной раскраски для смежных с ней вершин использовано не более $\Delta(G)$ цветов, значит, хотя бы один из цветов $\{1,2,\dots,\Delta(G)+1\}$ свободен и может быть применен. Отсюда следует еще одно неравенство для хроматического числа:

$$\chi(G) \leq \Delta(G) + 1,$$

причем алгоритм последовательной раскраски в любом случае использует не более $\Delta(G) + 1$ цветов.

Раскраска планарных графов. В XX веке одной из наиболее известных математических проблем была проблема четырех красок. Она возникла в середине XIX века и первоначально звучала как задача о раскраске географических карт, а позднее была переформулирована на язык теории графов. Требовалось доказать (или опровергнуть), что всякий планарный граф допускает правильную раскраску вершин в не более чем четыре цвета, т.е., что $\chi(G) \leq 4$ для планарного графа. Доказательство было получено только в 1976 г. Проблема была сведена к рассмотрению конечного (но очень большого) множества частных случаев, которые затем были исследованы с помощью компьютерных программ. Это первый в истории пример решения математической проблемы, полученного с помощью компьютера, которое человек не в состоянии проверить «вручную». Впоследствии доказательство было несколько упрощено, но ручная проверка остается все еще невозможной.

Приведем простое доказательство того, что для любого планарного графа достаточно пяти красок.

Теорема 8.1 (о пяти красках). *Для любого планарного графа G выполняется неравенство $\chi(G) \leq 5$.*

Доказательство. Сначала докажем, что в любом планарном графе имеется вершина степени не более 5. Действительно, если такой вершины нет, то степень каждой вершины не менее 6 и сумма степеней всех вершин графа не меньше $6n$. Но эта сумма равна удвоенному числу ребер, значит, число ребер в графе не меньше $3n$. Это противоречит следствию из теоремы 3.9, согласно которому это число не превосходит $3n - 6$.

Далее проводим индукцию по числу вершин. Пусть a – вершина наименьшей степени в плоском графе G. Если эту вершину удалить, то оставшийся граф G' по предположению индукции может быть раскрашен в 5 цветов. Если степень вершины a не больше 4, то для смежных с ней вершин используется не больше четырех цветов, значит, среди пяти цветов имеется цвет, в который можно окрасить вершину a. Пусть степень вершины a равна 5. Среди вершин в ее окрестности имеется пара несмежных вершин, иначе эта окрестность порождала бы граф K_5, который

не планарен. Пусть b и c – несмежные между собой вершины, смежные с a. В графе G' эти вершины принадлежат границе одной грани и можно выполнить слияние этих двух вершин так, что получится плоский граф G'', в котором вершины b и c заменены одной вершиной x. По предположению индукции этот граф допускает правильную раскраску в пять цветов, при этом для вершины x и трех оставшихся вершин из окрестности вершины a используется не более четырех цветов. Эту раскраску можно перенести на граф G, окрасив вершины b и c в цвет, который использовался для вершины x, и сохранив цвета остальных вершин. В окрестности вершины a теперь используется не более четырех цветов и имеется цвет, допустимый для a. □

8.2. Раскраска ребер

Наряду с задачей о раскраске вершин известна задача о *раскраске ребер* графа, когда цвета назначаются ребрам. Раскраска ребер называется правильной, если любые два ребра, имеющие общую вершину, окрашены в разные цвета. Минимальное число цветов, необходимое для правильной раскраски ребер графа G, называется *хроматическим индексом* (или *реберным хроматическим числом*) графа и обозначается через $\chi'(G)$.

В правильной раскраске ребер множество ребер одного цвета образует паросочетание. Поэтому такую раскраску можно трактовать как разбиение множества ребер графа на паросочетания.

Обозначим через $\Delta(G)$ максимальную степень вершины в графе G. При правильной реберной раскраске все ребра, инцидентные одной вершине, должны иметь разные цвета. Отсюда следует, что для любого графа выполняется неравенство $\chi'(G) \geq \Delta(\mathrm{G})$. Для некоторых графов имеет место строгое неравенство, например, $\Delta(C_3) = 2$, а $\chi'(C_3) = 3$. Оказывается, $\chi'(G)$ может отличаться от $\Delta(G)$ не более чем на 1. Приведем эту теорему без доказательства.

Теорема 8.2 (теорема Визинга). *Для любого графа G справедливы неравенства* $\Delta(G) \leq \chi'(G) \leq \Delta(\mathrm{G}) + 1$.

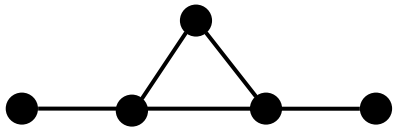
Из доказательства теоремы Визинга можно извлечь алгоритм, который за полиномиальное время находит раскраску ребер графа в не более чем $\Delta(G) + 1$ цвет. Однако выяснить, оптимальна ли эта раскраска (т.е. нельзя ли раскрасить ребра в $\Delta(G)$ цветов), является трудной (неподдающейся) задачей.

Задачи

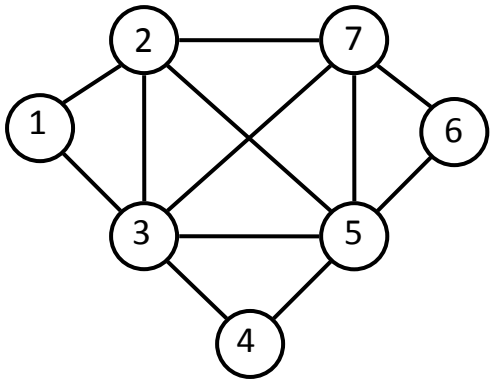
8.1. Найдите хроматические числа графов $K_{p,q}$, P_n, C_n, Q_n.

8.2. Найдите хроматическое число графа 1) $C_4 + C_7$; 2) $C_4 \circ C_7$; 3) $\overline{C_7}$.

8.3. Примените алгоритм раскрашивания, основанный на дереве решений, к графу, показанному на рисунке.



8.4. Примените алгоритм последовательной раскраски к графу, изображенному на рисунке, если вершины упорядочиваются а) по возрастанию номеров; б) по убыванию номеров.



8.5. Для каких из следующих графов алгоритм последовательной раскраски дает точный результат при любом упорядочении вершин: P_3, P_4, C_4, C_6, $K_{5,5}$?

8.6. Какие из следующих равенств справедливы для любых графов G_1 и G_2?
1) $\chi(G_1 + G_2) = \chi(G_1) + \chi(G_2)$;
2) $\chi(G_1 + G_2) = \max\{\chi(G_1), \chi(G_2)\}$;
3) $\chi(G_1 \circ G_2) = \chi(G_1) + \chi(G_2)$;
4) $\chi(G_1 \circ G_2) = \chi(G_1)\chi(G_2)$;
5) $\chi(G_1 \times G_2) = \chi(G_1)\chi(G_2)$.

8.7. Найдите хроматический индекс графов C_6, C_7, $K_{3,3}$, $K_{4,4}$, K_4, K_5.

8.8**. Разработайте алгоритм нахождения оптимальной раскраски вершин для графов интервалов (см. задачу 1.36).

9. Потоки

В прикладных задачах графы обычно появляются не в чистом виде, а как части более сложных моделей. При этом часто вершинам и/или ребрам графа приписываются какие-нибудь числа, они могут означать расстояния, длительности, стоимости и т.д. В этой и следующих главах будут рассмотрены задачи на графах, у которых числа приписаны ребрам.

9.1. Задача о максимальном потоке

В этой главе будем рассматривать ориентированные графы без петель и кратных ребер. Для вершины x множество всех входящих в нее ребер обозначаем через $E^+(x)$, а множество выходящих – через $E^-(x)$.

Сетью называется орграф, в котором

1) каждому ребру e приписано положительное число $c(e)$, называемое *пропускной способностью* ребра;
2) выделены две вершины s и t, называемые соответственно *источником* и *стоком*, при этом нет ребер, входящих в источник и ребер, выходящих из стока, т.е. $E^+(s) = E^-(t) = \emptyset$.

Вершины сети, отличные от источника и стока, будем называть *внутренними*.

Пусть задана сеть N с множеством вершин V и множеством ребер E. Пусть f – функция с вещественными значениями, определенная на множестве E. Для вершины x обозначим

$$f^+(x) = \sum_{e \in E^+(x)} f(e),$$

$$f^-(x) = \sum_{e \in E^-(x)} f(e).$$

Функция f называется *потоком* в сети N, если она удовлетворяет условиям:

(1) $0 \le f(e) \le c(e)$ для каждого ребра e;

(2) $f^+(x) = f^-(x)$ для каждой внутренней вершины x.

На рисунке 9.1 показан пример сети и потока в ней. В дроби, приписанной каждому ребру, знаменатель представляет пропускную способность ребра, а числитель – величину потока на этом ребре.

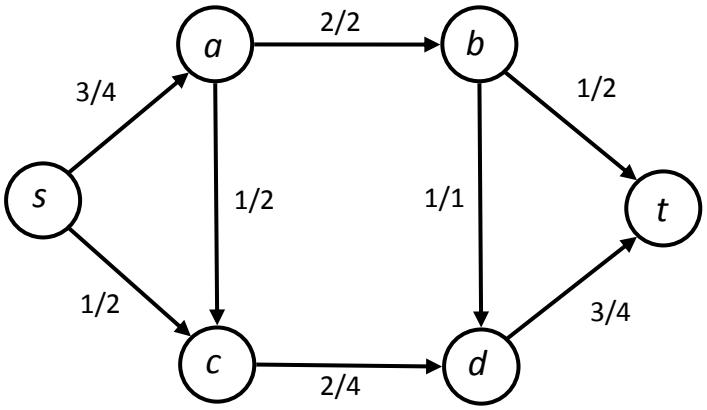


Рис. 9.1. Сеть и поток в ней

Условие (2) называется *условием сохранения потока*. Так как каждое ребро является входящим для одной вершины и выходящим для другой, то суммарный входящий поток по всем вершинам равен суммарному выходящему потоку:

$$\sum_{x \in V} f^+(x) = \sum_{x \in V} f^-(x).$$

Из этого равенства и условия сохранения потока следует, что

$$f^-(s) = f^+(t),$$

т.е. суммарный поток из источника равен суммарному потоку в сток. Величина $f^-(s)$ обозначается через $|f|$ и называется *величиной потока*. В примере на рисунке $|f| = 4$. *Задача о максимальном потоке* состоит в том, чтобы для данной сети найти поток наибольшей величины.

Введем некоторые вспомогательные понятия и установим два полезных соотношения. Пусть $X \subseteq V$, $\bar{X} = V - X$. Множество всех ребер, у которых начальная вершина принадлежит множеству X, а конечная – множеству $\bar{X}$, будем обозначать через $(X, \bar{X})$. Множество $(X, \bar{X})$ называется *разрезом* сети, если $s \in X$, $t \in \bar{X}$. Если f – поток, то для каждого множества $X \subseteq V$ можно определить суммарный поток из множества X:

$$f(X) = \sum_{e \in (X, \bar{X})} f(e).$$

Лемма 1. *Для любого потока f и любого разреза $(X, \bar{X})$ выполняется равенство* $|f| = f(X) - f(\bar{X})$.

Доказательство. Рассмотрим величину

$$S = \sum_{x \in X} f^-(x) - \sum_{x \in X} f^+(x).$$

В множестве X все вершины внутренние, кроме источника s. Поэтому ввиду сохранения потока во внутренних вершинах все слагаемые в S уничтожаются, кроме одного:

$$S = f^-(s) = |f|.$$

С другой стороны, рассмотрим вклад, вносимый в эту сумму некоторым ребром $e = (x, y)$. Он зависит от того, в каком отношении к множеству X находится это ребро. Рассмотрим все возможные варианты.

1) Если $x \in X$, $y \in X$, то величина $f(e)$ присутствует в двух слагаемых суммы S: в $f^-(x)$ со знаком плюс и в $f^+(y)$ со знаком минус, так что общий вклад ребра e в этом случае равен 0.

2) Если $x \in X$, $y \in \bar{X}$, (то есть $e \in (X, \bar{X})$), то $f(e)$ присутствует только в слагаемом $f^-(x)$ со знаком плюс, вклад ребра e равен $+f(e)$.

3) Если $x \in \bar{X}$, $y \in X$, (то есть $e \in \bar{X}, X)$), то $f(e)$ присутствует только в слагаемом $f^+(y)$ со знаком минус; вклад ребра e равен $-f(e)$;

4) Если $x \in \bar{X}$, $y \in \bar{X}$, то ребро e не представлено ни водном слагаемом из S и его вклад равен 0.

Отсюда следует, что

$$S = \sum_{e \in (X, \bar{X})} f(e) - \sum_{e \in (\bar{X}, X)} f(e) = f(X) - f(\bar{X}).$$

□

Пропускной способностью множества $(X, \bar{X})$ называется сумма пропускных способностей всех его ребер, она обозначается через $c(X)$. Очевидно, для любого множества $X \subseteq V$ и любого потока f выполняется неравенство $f(X) \leq c(X)$.

Лемма 2. *Для любого потока f и любого разреза $(X, \bar{X})$ имеет место неравенство $|f| \leq c(X)$.*

Доказательство. Это следует из леммы 1 и неравенств $f(X) \leq c(X)$, $f(\bar{X}) \geq 0$.

9.2. Метод увеличивающих путей

Поток на рисунке 9.1 не является максимальным – его можно увеличить, например, добавив по единице на ребрах пути s, a, c, d, t. Получится поток величины 5, показанный на рисунке 9.2 вверху. Но и он

не максимален. Можно увеличить поток на единицу на ребрах (s,c), (c,d), (b,t) и уменьшить на единицу на ребре (b,d). Иначе говоря, вдоль неориентированного пути s,c,d,b,t (показан пунктиром) поток увеличивается на всех ребрах, направление которых совпадает с направлением пути, и уменьшается на ребре противоположного направления. Условие сохранения останется выполненным, а величина потока станет равной 6 (рисунок 9.2, внизу).

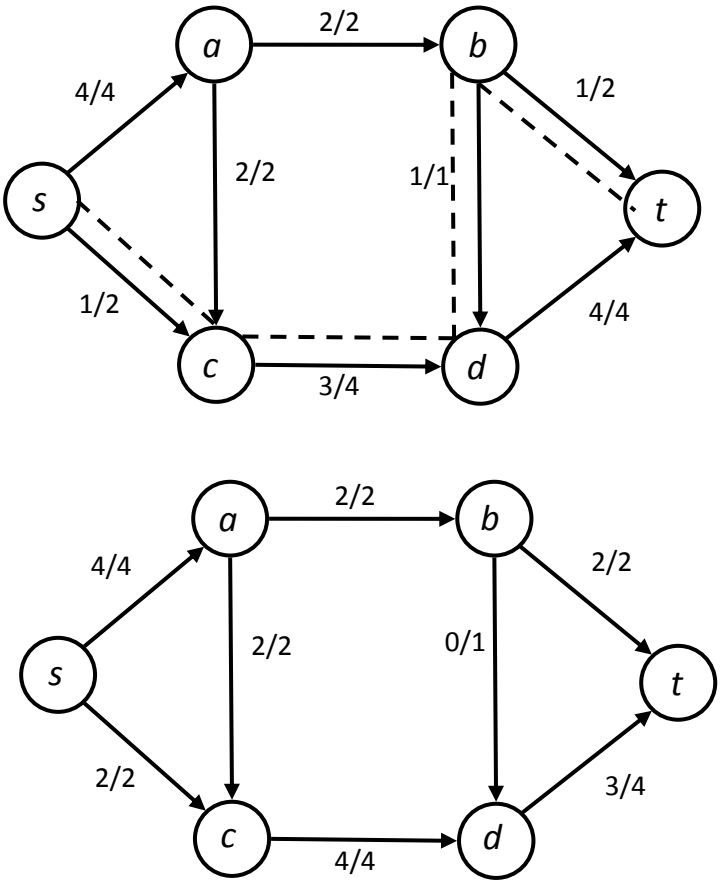


Рис. 9.2. Увеличение потока

Приведенный пример иллюстрирует общий метод, на котором основаны многие алгоритмы решения задачи о максимальном потоке – метод увеличивающих путей. Он является обобщением метода чередующихся путей для задачи о паросочетании.

Допустим, в сети N задан поток f. Пусть $x_1, e_1, x_2, e_2, \dots, e_{k-1}, x_k$ – неориентированный путь в сети. Ребро e_i назовем *прямым* ребром этого пути, если $e_i = (x_i, x_{i+1})$, и *обратным*, если $e_i = (x_{i+1}, x_i)$. Путь назовем *подходящим* относительно потока f, если для каждого прямого ребра выполняется неравенство $f(e_i) < c(e_i)$, а для каждого обратного – неравенство $f(e_i) > 0$. Таким образом, на каждом прямом ребре подходящего пути поток можно увеличить, а на каждом обратном – уменьшить. *Увеличивающий путь* – это подходящий путь из источника в сток.

Лемма 3. *Если относительно потока f имеется увеличивающий путь, то этот поток не максимален.*

Доказательство. Пусть P – увеличивающий путь для потока f, A – множество всех прямых, B – множество всех обратных ребер этого пути. Положим

$$\delta_1 = \min_{e \in A} \{c(e) - f(e)\},$$

$$\delta_2 = \min_{e \in B} f(e).$$

Тогда на каждом прямом ребре пути можно увеличить поток на величину δ_1, а на каждом обратном – уменьшить на величину δ_2. Возьмем $\delta = \min\{\delta_1, \delta_2\}$ и определим на ребрах сети новую функцию f':

$$f'(e) = \begin{cases} f(e) + \delta, & \text{если } e \text{ прямое,} \\ f(e) - \delta, & \text{если } e \text{ обратное,} \\ f(e), & \text{если } e \text{ не принадлежит пути } P. \end{cases}$$

Легко видеть, что условия (1) и (2) из определения потока для функции f' выполняются, так что эта функция является потоком. Вместе с тем, так как у источника есть только выходящие ребра, то первое ребро пути P – прямое, поэтому $|f'| = |f| + \delta$, причем $\delta > 0$. □

Теорема 9.1 (об увеличивающем пути). *Поток максимален тогда и только тогда, когда относительно него нет увеличивающего пути.*

Доказательство. Если поток максимален, то из леммы 3 следует, что увеличивающего пути нет. Обратно, пусть f – поток, относительно которого нет увеличивающего пути. Покажем, что этот поток максимален. Для этого рассмотрим множество X, состоящее из всех вершин сети, достижимых подходящими путями из вершины s. Так как увеличивающих путей нет, то $t \in \bar{X}$, так что $(X, \bar{X})$ – разрез. Пусть $e = (x, y)$ – ребро этого разреза. Вершина x достижима из источника подходящим путем, а вершина y недостижима. Тогда $f(e) = c(e)$, так как иначе к подходящему пути, ведущему из s в x можно было бы добавить ребро e и вершину y, получился бы подходящий путь из s в y. Итак, на каждом ребре разреза $(X, \bar{X})$ поток равен пропускной способности, следовательно, $f(X) = c(X)$. Аналогично, рассматривая ребра из множества $(\bar{X}, X)$, убеждаемся, что поток на каждом таком ребре равен 0, в противном случае опять можно было бы продолжить некоторый подходящий путь до вершины из $\bar{X}$. Следовательно, $f(\bar{X}) = 0$. Применяя лемму 1, получаем

$$|f| = f(X) - f(\bar{X}) = c(X).$$

По лемме 2 никакой поток не может иметь величину больше $c(X)$. Значит, f – максимальный поток. □

Теорема 9.2 (о максимальном потоке и минимальном разрезе). *Максимальная величина потока в сети равна минимальной пропускной способности разреза этой сети.*

Доказательство. Из доказательства теоремы 9.1 видно, что если f – максимальный поток в некоторой сети, то в этой сети существует такой разрез $(X, \bar{X})$, что $|f| = c(X)$. Но тогда этот разрез имеет минимальную пропускную способность среди всех разрезов данной сети. Действительно, если существует разрез $(X', \overline{X'})$ с меньшей пропускной сполсобностью: $c(X') < c(X)$, то получается $|f| > c(X')$, а это противоречит лемме 2. □

Метод увеличивающих путей для задачи о максимальном потоке разработали Форд и Фалкерсон в 1955 г. Известно несколько алгоритмов, реализующих этот метод, они различаются, в частности, стратегией поиска увеличивающих путей. Первый алгоритм, для которого была получена верхняя оценка трудоемкости, предложили Эдмондс и Карп в 1972 г. В этом алгоритме всегда ищется кратчайший (по числу ребер) увеличивающий путь. Удобно этот поиск вести не на исходной сети N, а на *остаточной сети* R, которая при заданном на сети N потоке f определяется следующим образом. Множество вершин, источник и сток у остаточной сети те же, что у исходной. Пусть e – ребро исходной сети. Тогда

1) если $f(e) < c(e)$, то ребро e включается в сеть R и ему в этой сети присваивается пропускная способность $c'(e) = c(e) - f(e)$;

2) если $f(e) > 0$ и ребро e соединяет две внутренние вершины, то к сети R добавляется ребро противоположного направления $\bar{e}$ с пропускной способностью $c'(\bar{e}) = f(e)$.

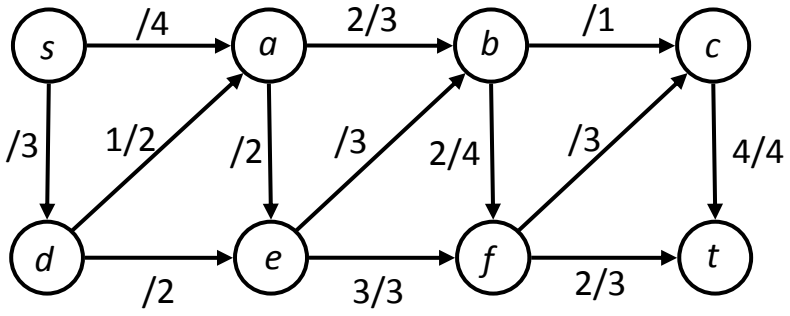
Легко видеть, что увеличивающие пути в исходной сети находятся во взаимно однозначном соответствии с ориентированными путями из источника в сток в остаточной сети. В алгоритме Эдмондса-Карпа нужно в остаточной сети искать кратчайший ориентированный путь из источника в сток. Это можно сделать за линейное время с помощью поиска в ширину. Если увеличивающий путь обнаружен, поток увеличивается. Для нового потока строится остаточная сеть и т.д., пока не будет построен поток, относительно которого нет увеличивающего пути (в остаточной сети нет ориентированного пути из источника в сток. Общая оценка трудоемкости алгоритма Эдмондса-Карпа оценивается как $O(m^2 n)$. В настоящее время известны и более быстрые алгоритмы для задачи о максимальном потоке.

Когда максимальный поток найден, нетрудно найти и минимальный разрез в сети, т.е. разрез с минимальной пропускной способностью. Нужно найти все вершины, достижимые из источника подходящими путями. Пусть X – множество всех таких вершин, тогда множество всех ребер из X в $\bar{X}$ является минимальным разрезом.

Применение к задаче о паросочетании. Алгоритм нахождения максимального потока можно применять для нахождения наибольшего паросочетания в двудольном графе. Пусть A и B – доли такого графа. Ориентируем все ребра графа по направлению от A к B. Добавим новую вершину s, объявим ее источником и добавим ребра из s ко всем вершинам доли A. Добавим сток t и ребра от всех вершин доли B к t. Припишем каждому ребру пропускную способность 1. Будем искать максимальный поток в построенной сети, используя метод увеличивающих путей, начиная с нулевого потока. Каждый раз, найдя увеличивающий путь, будем увеличивать поток вдоль этого пути на максимально возможное значение (в данном случае это значит – на 1). Тогда в итоговом потоке ребра, на которых поток равен 1 (а на остальных он равен 0), образуют систему путей из s в t, не имеющих общих вершин, кроме s и t. Средние ребра этих путей образуют наибольшее паросочетание.

Задачи

9.1. На рисунке показана сеть (знаменатели дробей – пропускные способности ребер). В этой сети задан поток, значения потока показаны на некоторых ребрах (числители дробей), но не на всех. Восстановите пропущенные значения.

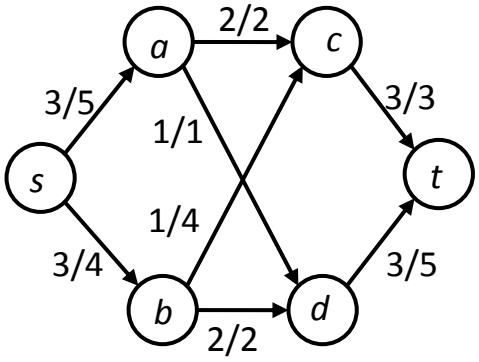


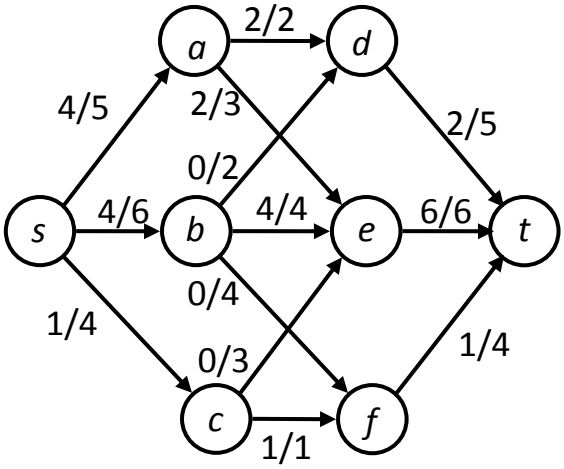
9.2. Пусть f_1 и f_2 – потоки в некоторой сети. Какие из следующих функций обязательно будут потоками в той же сети?
1) $f_1 + f_2$; 2) $\frac{1}{2}(f_1 + f_2)$; 3) $\frac{1}{3}(f_1 + f_2)$; 4) $\frac{2}{3}(f_1 + f_2)$;
5) $\frac{1}{3}f_1 + \frac{2}{3}f_2$; 6) $f_1 - f_2$; 7) $f_1 f_2$; 8) $\sqrt{f_1 f_2}$.

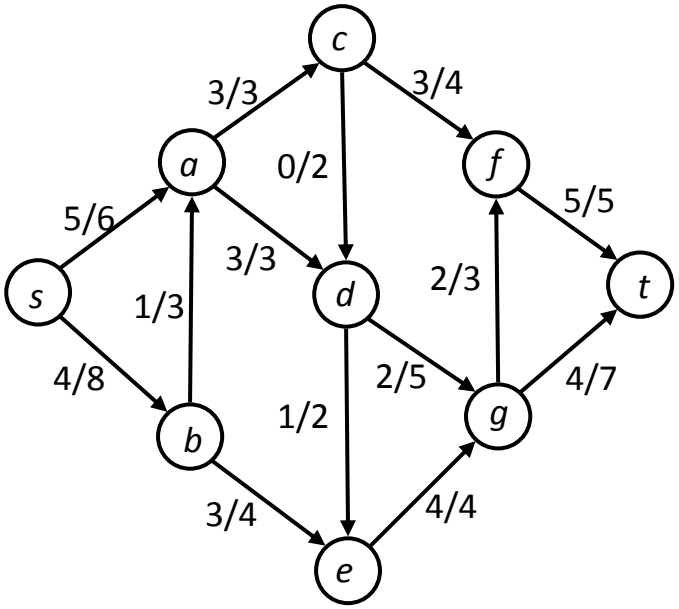
9.3. Докажите, что если f_1 и f_2 – потоки в некоторой сети, то при любом α, $0 \le \alpha \le 1$, функция $\alpha f_1 + (1-\alpha)f_2$ является потоком в той же

сети.

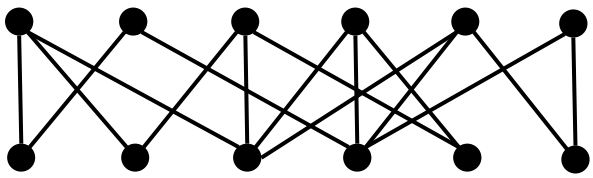
9.4. Выясните, является ли показанный на рисунке поток максимальным. Если да, то найдите минимальный разрез в этой сети. Если нет, то найдите максимальный поток.







9.5. Найдите наибольшее паросочетание в данном графе с помощью алгоритма для задачи о максимальном потоке.



10. Оптимальные каркасы

В задаче об оптимальном каркасе (она известна также как задача о кратчайшем остовном дереве) задан обыкновенный граф $G = (V, E)$, каждому ребру которого приписано число, называемое весом ребра. Вес ребра (x, y) будем обозначать через $w(x, y)$. Вес множества $X \subseteq E$ определяется как сумма весов составляющих его ребер и обозначается через $w(X)$. Требуется в графе G найти каркас минимального веса. Для решения этой задачи известно несколько алгоритмов. Рассмотрим два из них.

10.1. Алгоритм Прима

Будем предполагать, что граф G связен, так что решением задачи всегда будет дерево. В алгоритме Прима на каждом шаге рассматривается частичное решение задачи, представляющее собой дерево. Вначале это дерево состоит из единственной вершины, в качестве которой может быть выбрана любая вершина графа. Затем к дереву последовательно добавляются ребра до тех пор, пока не получится остовное дерево, т.е. каркас. Необходимо следить, чтобы при добавлении каждого нового ребра опять получалось дерево. Это значит, что новое ребро не может соединять двух вершин, уже принадлежащих дереву, так как в этом случае образуется цикл. Новое ребро также не может соединять двух вершин, не принадлежащих дереву, так как в результате получится несвязный подграф. Значит, новое ребро должно соединять вершину дерева с вершиной, еще не принадлежащей дереву. Такие ребра будем называть *подходящими* относительно рассматриваемого дерева. В алгоритме Прима применяется следующее правило выбора: на каждом шаге из всех подходящих ребер выбирается ребро наименьшего веса. Это ребро вместе с одной новой вершиной добавляется к дереву. Для формального описания обозначим строящееся дерево через F.

Алгоритм Прима.

1 Выбрать произвольную вершину a;
2 создать дерево F из одной вершины a;
3 повторить $n - 1$ раз
4 найти ребро наименьшего веса (x, y) среди всех ребер, у которых $x \in F, y \notin F$;
5 добавить к F ребро (x, y).

На рисунке 9.3 показана работа алгоритма Прима на примере. Вначале дерево состоит из единственной вершины 1, затем к нему

последовательно добавляются ребра (1,2), (2,6), (2,3), (3,4), (5,6), (3,8), (8,7).

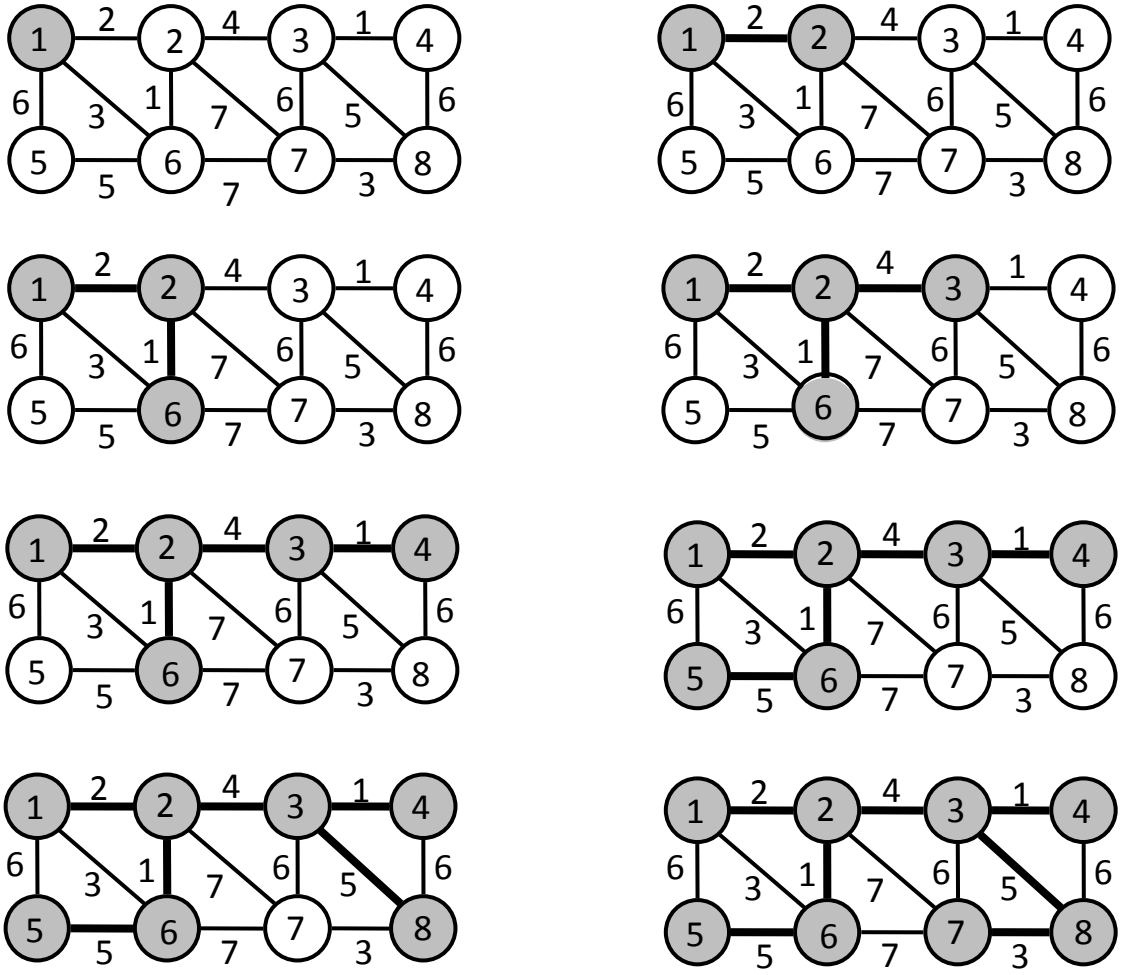


Рис. 9.3. Алгоритм Прима

Докажем, что алгоритм Прима действительно находит оптимальный каркас.

Теорема 10.1. *Каркас, построенный с помощью алгоритма Прима, имеет наименьший вес среди всех каркасов данного графа*.

Доказательство. Дерево, которое является подграфом какого-нибудь оптимального каркаса, будем называть *частичным оптимальным каркасом,* сокращенно ЧОК. Покажем, что каждое дерево, построенное после нескольких шагов алгоритма Прима, является ЧОК. Отсюда следует, что окончательный результат работы алгоритма будет оптимальным каркасом.

Дерево, состоящее из одной (любой) вершины – ЧОК, так как любой каркас содержит все вершины графа. Покажем, что свойство дерева быть

ЧОК сохраняется при каждом добавлении к дереву нового ребра в соответствии с алгоритмом.

Допустим, к некоторому моменту работы алгоритма построено дерево F и оно является ЧОК. Пусть T – оптимальный каркас, содержащий F в качестве подграфа, $e = (x, y)$ – подходящее ребро минимального веса относительно дерева F. Обозначим дерево, получающееся добавлением ребра e к дереву F, через F'. Нужно доказать, что F' – ЧОК.

Если ребро e принадлежит T, то F' – подграф дерева T и, следовательно, ЧОК. Допустим, e не принадлежит T. На пути между вершинами x и y в дереве T есть хотя бы одно подходящее ребро (так как одна из вершин x, y принадлежит дереву F, а другая не принадлежит). Пусть e' – такое ребро. Рассмотрим подграф T', получающийся из T удалением ребра e' и добавлением ребра e. При удалении ребра e' связность в подграфе T нарушается – вершины x и y оказываются в разных компонентах, но при добавлении ребра e она восстанавливается. Значит, T' – тоже дерево. Так как $w(e) \leq w(e')$, то $w(T') \leq w(T)$. Но T – оптимальный каркас, поэтому $w(T')$ не может быть меньше, чем $w(T)$. Следовательно, $w(T') = w(T)$ и T' – тоже оптимальный каркас. А так как F' – подграф графа T', то F' – ЧОК. □

Оценим время работы алгоритма Прима в случае, когда заданный граф G – полный. Рассмотрим момент работы алгоритма, когда построенный частичный каркас содержит k вершин. В этот момент имеется $k(n-k)$ подходящих ребер, среди которых нужно выбрать ребро наименьшего веса. Число k меняется от 1 до $n-1$, так что всего придется рассмотреть

$$\sum_{k=1}^{n-1} k(n-k) = n\sum_{k=1}^{n-1} k - \sum_{k=1}^{n-1} k^2 = \frac{n^2(n-1)}{2} - \frac{(n-1)n(2n-1)}{6} = \frac{n^3-n}{6}$$

пар. Таким образом, трудоемкость алгоритма будет $O(n^3)$.

Небольшое усовершенствование позволяет на порядок ускорить этот алгоритм. Допустим, что для каждой вершины x, не принадлежащей текущему дереву F, известна такая вершина $b(x)$ из F, что ребро $(x, b(x))$ имеет наименьший вес среди всех ребер вида (x, y), где $y \in F$. Тогда, если F содержит k вершин, для поиска подходящего ребра наименьшего веса достаточно будет сравнить не $k(n-k)$, а только $n-k$ ребер вида $(x, b(x))$, а общее число анализируемых ребер будет равно

$$\sum_{k=1}^{n-1} (n-k) = \frac{n(n-1)}{2}.$$

В этом случае, однако, необходимы дополнительные действия для обновления таблицы значений функции $b(x)$ при добавлении новой вершины к дереву F. Сначала, когда это дерево состоит из единственной вершины a, полагаем $b(x) = a$ для всех $x \in V - \{a\}$. В дальнейшем эти значения могут меняться. Допустим, на некотором шаге к дереву F присоединяется вершина c. Тогда для каждой вершины $x \notin F$ либо сохраняется старое значение $b(x)$, либо устанавливается новое $b(x) = c$, в зависимости от того, какое из ребер $(x, b(x))$ и (x, c) имеет меньший вес. По окончании работы алгоритма массив b будет массивом отцов построенного дерева относительно корня a. Поэтому отпадает необходимость отдельно запоминать добавляемые к дереву ребра. Алгоритм теперь можно записать следующим образом.

Алгоритм Прима усовершенствованный.

1 Выбрать произвольную вершину a;
2 создать дерево F из одной вершины a;
3 **for** $x \in V - \{a\}$ **do** положить $b(x) = a$**;**
4 **for** $k = 1$ **to** $n - 1$ **do**
5 найти вершину $x \notin F$ с наименьшим значением $w(x, b(x))$;
6 добавить к F ребро $(x, b(x))$;
7 **for** $y \notin F$ **do if** $w(y, b(y)) > w(y, x)$ **then** положить $b(y) = x$.

При каждом k цикл в строке 7 повторяется $n - k$ раз. Таким образом, дополнительное время, затрачиваемое на обслуживание массива b, тоже оценивается сверху квадратичной функцией и общая оценка трудоемкости усовершенствованного алгоритма Прима будет $O(n^2)$.

10.2. Алгоритм Краскала

Другой алгоритм для задачи об оптимальном каркасе известен как алгоритм Краскала (J. Kruskal). В нем тоже на каждом шаге рассматривается частичное решение. Отличие от алгоритма Прима состоит в том, что в алгоритме Краскала частичное решение представляет собой остовный лес F графа G, т.е. лес, состоящий из всех вершин графа и некоторых его ребер. Вначале F не содержит ни одного ребра, т.е. состоит из изолированных вершин. Затем к нему последовательно добавляются ребра, пока не будет построен каркас графа G. Пусть F – лес, построенный к очередному шагу. Новое ребро можно добавить к этому лесу так, чтобы он остался лесом, только если оно соединяет вершины из разных компонент связности леса F. Теперь именно такие ребра будем называть подходящими. Для выбора добавляемого ребра применяется тот же принцип, что и в алгоритме Прима – из всех подходящих ребер выбирается

ребро наименьшего веса. Для того чтобы облегчить поиск этого ребра, вначале все ребра графа упорядочиваются по возрастанию весов: $w(e_1) \leq \cdots \leq w(e_m)$. Теперь последовательность ребер $e_1, \ldots, e_m$ достаточно просмотреть один раз и для очередного рассматриваемого ребра нужно определить, является ли оно подходящим относительно построенного к этому моменту леса F. Если ребро подходящее, оно добавляется к F , в противном случае просто пропускается.

На рисунке 9.4 работа алгоритма Краскала демонстрируется на примере.

Доказательство того, что алгоритм Крускала действительно строит каркас минимального веса похоже на аналогичное доказательство для алгоритма Прима и здесь не приводится.

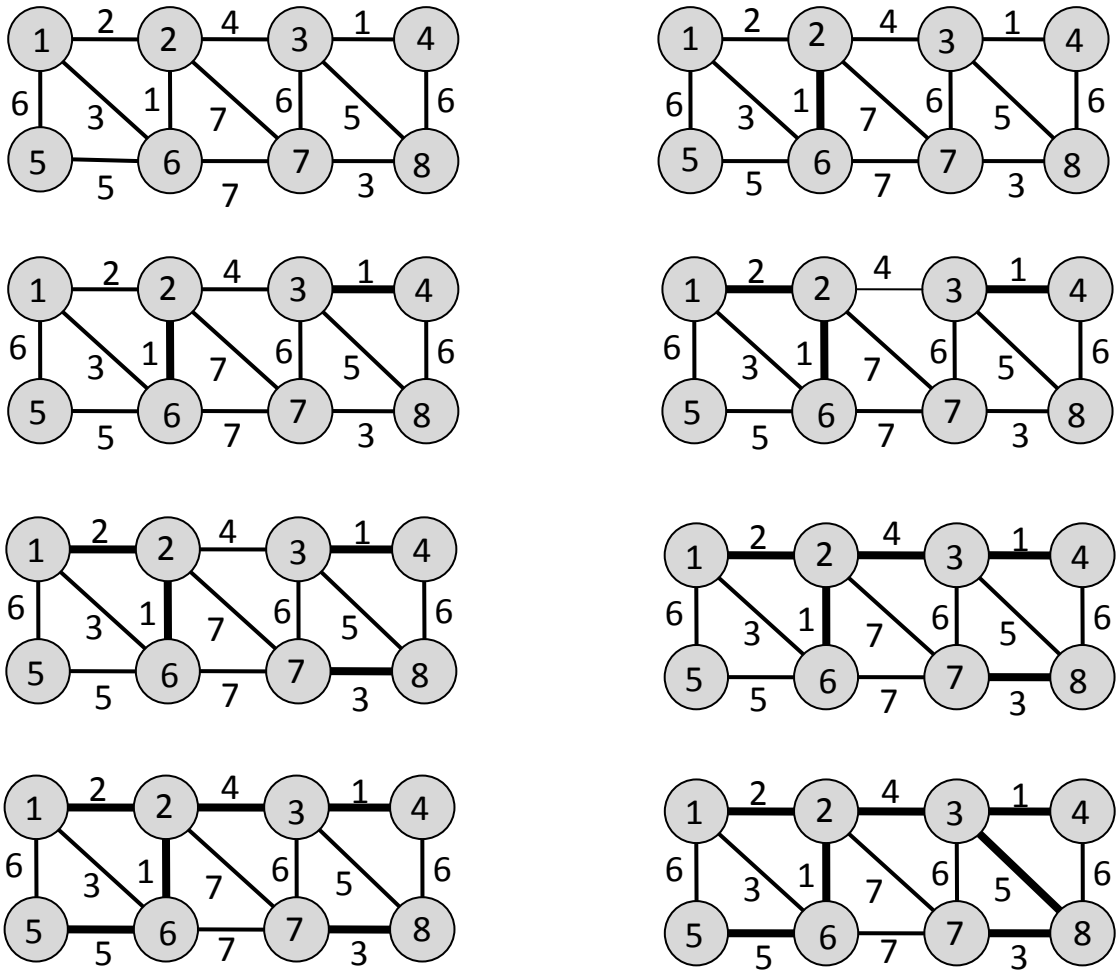


Рис. 9.4. Алгоритм Краскала

Скорость работы алгоритма Краскала зависит от того, как организовано хранение информации о текущем лесе и как выполняется проверка подходящих ребер. Применение специальных структур данных, которые мы не будем здесь рассматривать, позволяет реализовать этот алгоритм (без учета предварительной сортировки ребер) с оценкой времени работы $O(mf(n))$, где $f(n)$ – очень медленно растущая функция.

В частности, $f(n) < 5$ для $n < 2^{65536}$. При такой реализации общая трудоемкость алгоритма определяется временем, затрачиваемым на предварительную сортировку, которое оценивается как $O(m \log m)$.

Задачи

10.1. Найдите оптимальный каркас по матрице весов ребер с помощью алгоритма Прима (прочерк означает отсутствие ребра).

$$\begin{pmatrix} - & 2 & 4 & 2 & - & - & - \\ 2 & - & 2 & 3 & 3 & - & 1 \\ 4 & 2 & - & 3 & - & 3 & 2 \\ 2 & 3 & 3 & - & - & 1 & 3 \\ - & 3 & - & - & - & - & 4 \\ - & - & 3 & 1 & - & - & 4 \\ - & 1 & 2 & 3 & 4 & 4 & - \end{pmatrix} \qquad \begin{pmatrix} - & 3 & 5 & 3 & 4 & - & 2 \\ 3 & - & 1 & - & 3 & 4 & 3 \\ 5 & 1 & - & 2 & 1 & 3 & - \\ 3 & - & 2 & - & - & 4 & 4 \\ 4 & 3 & 1 & - & - & 2 & 2 \\ - & 4 & 3 & 4 & 2 & - & 5 \\ 2 & 3 & - & 4 & 2 & 5 & - \end{pmatrix}$$

10.2. Найдите оптимальный каркас с помощью алгоритма Краскала.

$$\begin{pmatrix} - & 4 & 3 & 2 & 3 & 2 & - & 1 \\ 4 & - & 1 & - & 1 & 5 & 3 & 4 \\ 3 & 1 & - & 3 & 2 & 3 & 5 & - \\ 2 & - & 3 & - & 4 & 2 & - & 2 \\ 3 & 1 & 2 & 4 & - & - & 3 & 4 \\ 2 & 5 & 3 & 2 & - & - & 2 & 3 \\ - & 3 & 5 & - & 3 & 2 & - & 2 \\ 1 & 4 & - & 2 & 4 & 3 & 2 & - \end{pmatrix}$$

10.3. Какие из следующих утверждений верны для любого графа с весами ребер?

1) Если в графе имеется единственное ребро наименьшего веса, то оно принадлежит каждому оптимальному каркасу.
2) Если в графе имеются точно два ребра наименьшего веса, то они оба принадлежат каждому оптимальному каркасу.
3) Если в графе имеются точно три ребра наименьшего веса, то все они принадлежат каждому оптимальному каркасу.
4) Каждое ребро минимального веса принадлежит какому-нибудь оптимальному каркасу.

10.4. Вершины полного графа K_{pq} размещаются в целочисленных точках прямоугольника $[1, p] \times [1, q]$. Вес каждого ребра равен евклидовой длине отрезка, соединяющего вершины этого ребра.

1) Чему равен вес оптимального каркаса для этого графа?
2) Каков будет вес оптимального каркаса, если из графа удалить все ребра длины 1?
3) Каков будет вес оптимального каркаса, если из графа удалить все ребра с длинами $1, 2$ и $\sqrt{2}$, а $p = q = 8$?
4) Каков будет вес оптимального каркаса, если из графа удалить все ребра с длинами 1 и $\sqrt{2}$, а $p = q = 8$?

11. Кратчайшие пути

В главе 4 была рассмотрена задача о кратчайших путях, в которой под длиной пути понималось число ребер в нем. Теперь рассмотрим более общую задачу, в которой длины ребер могут быть различными. Предполагаем, что задан связный неориентированный граф $G = (V, E)$ и для каждого его ребра (x, y) указана длина этого ребра $l(x, y)$ – положительное число. Длина пути есть сумма длин его ребер, длину пути P обозначаем через $l(P)$. Длина пути, состоящего из одной вершины, считается равной 0.

Рассмотрим задачу отыскания путей наименьшей длины от заданной вершины a до всех остальных вершин графа. Наиболее известным способом решения этой задачи является описываемый далее алгоритм Дейкстры (E. Dijkstra).

Решение задачи о кратчайших путях из заданной вершины a удобно представлять в виде *дерева кратчайших путей.* Это дерево с корнем a, являющееся каркасом графа G, в котором путь от любой вершины до корня является кратчайшим путем между этими вершинами во всем графе.

Алгоритм Дейкстры очень похож на алгоритм Прима. В нем процесс построения дерева тоже начинается с одной вершины, только в алгоритме Прима это могла быть любая вершина графа, а теперь это должна быть вершина a. В дальнейшем на каждом шаге к дереву присоединяется одно новое ребро (и одна вершина). Это ребро выбирается из подходящих ребер, причем понятие подходящего ребра здесь такое же – это ребро, соединяющее вершину дерева с вершиной, ему не принадлежащей. И правило выбора добавляемого ребра такое же – среди подходящих ребер выбирается ребро наименьшего веса. Разница в том, что в алгоритме Прима веса ребер заданы изначально, а в алгоритме Дейкстры они вычисляются.

Пусть T – частичное дерево, построенное к некоторому моменту работы алгоритма. На множестве вершин графа определим функцию $L(x)$ (при описании алгоритма Дейкстры ее значения обычно называют метками вершин). Для каждой вершины x дерева T значение $L(x)$ равно длине пути между вершинами x и a в дереве T (позже покажем, что это равно длине кратчайшего пути между x и a во всем графе). Для вершины x, не принадлежащей дереву, значение $L(x)$ равно минимальной длине пути из a в x, в котором все ребра принадлежат дереву, кроме последнего. Иначе говоря, для $x \notin T$

$$L(x) = \min_{y \in T}\{L(y) + l(x, y)\}.$$

Значение y, на котором достигается этот минимум, обозначим через $p(x)$. Величину $L(x)$ можно рассматривать как оценку длины кратчайшего пути из a в x, учитывающую только известные уже кратчайшие пути к

вершинам дерева T, а $p(x)$ – как потенциального отца вершины x в дереве кратчайших путей.

Алгоритм действует так: на очередном шаге среди вершин, не принадлежащих текущему дереву, выбирается вершина x с наименьшей меткой $L(x)$ и к дереву добавляется ребро $(x, p(x))$. Затем значения функций L и p пересчитываются с учетом изменившегося дерева.

Алгоритм Дейкстры.

```
1  Создать дерево T из одной вершины a;
2  for x ∈ V  do положить p(x) = a, L(x) = l(a, x);
3  for k = 1 to n − 1 do
4      найти вершину x ∉ T с наименьшим значением L(x);
5      добавить к T ребро (x, p(x));
6      for y ∉ T do
7          if  L(x) + l(x, y) < L(y)
8          then положить p(y) = x, L(y) = L(x) + l(x, y).
```

Теорема 11.1. *Дерево, построенное с помощью алгоритма Дейкстры, является деревом кратчайших путей.*

Доказательство. Докажем, что после каждого повторения основного цикла (строки 4-8) в построенном дереве T каждый путь от какой-нибудь вершины к корню является кратчайшим путем между этими вершинами в графе. Это верно вначале, когда дерево состоит из одной вершины. Покажем, что это свойство сохраняется при добавлении к дереву вершины x и ребра $(x, p(x))$ в соответствии с алгоритмом (строки 4-5). Нужно доказать, что путь от вершины a до вершины x в обновленном дереве T является кратчайшим путем в графе. Обозначим этот путь через P. Его длина $l(P) = L(x)$. Допустим, в графе имеется более короткий путь P' между вершинами a и x, т.е. $l(P') < l(P)$. Пусть (u, v) – первое (считая от вершины a) ребро пути P', у которого $u \in T$, $v \notin T$. Рассмотрим отрезок пути P' от вершины a до вершины v, обозначим этот отрезок через P''. Так как длины ребер неотрицательны и путь P'' является частью пути P', то $l(P'') \leq l(P')$. Но P'' – это путь из a в v, в котором все ребра принадлежат дереву, кроме последнего. По определению функции L минимальная длина такого пути равна $L(v)$. Значит, $L(v) \leq l(P'')$. Получаем цепочку соотношений

$$L(v) \leq l(P'') \leq l(P') < l(P) = L(x),$$

т.е. $L(v) < L(x)$, а это противоречит выбору вершины x. □

Таким образом, по окончании работы алгоритма значение $L(x)$ для каждой вершины x – это длина кратчайшего пути из a в x, а $p(x)$ – отец вершины x в дереве кратчайших путей.

Трудоемкость алгоритма Дейкстры оценивается так же, как трудоемкость усовершенствованного алгоритма Прима, т.е. $O(n^2)$.

Задачи

11.1. По матрице длин ребер графа с помощью алгоритма Дейкстры найдите

1) кратчайшие пути от вершины 7 до всех остальных вершин;
2) кратчайший путь между вершинами 1 и 4.

$$\begin{pmatrix} 0 & 3 & - & - & - & - & 2 \\ 3 & 0 & 2 & - & 6 & - & - \\ - & 2 & 0 & 2 & - & 1 & - \\ - & - & 2 & 0 & 5 & 5 & - \\ - & 6 & - & 5 & 0 & 1 & - \\ - & - & 1 & 5 & 1 & 0 & 1 \\ 2 & - & - & - & - & 1 & 0 \end{pmatrix}$$

11.2. Каркасы, построенные для некоторого графа с помощью алгоритмов Прима, Краскала и Дейкстры, имеют соответственно веса a, b и c. Какое из следующих соотношений обязательно выполняются для этих чисел?

1) $a \geq c$; 2) $a = b$; 3) $b \leq c$; 4) $b = c$.

Ответы

1.3. 20.
1.4. 12.
1.5. 11 графов.
1.6. а) 4 графа; б) таких графов не существует.
1.8. $\frac{1}{n-2}\sum_{i=1}^{n} m_i$.
1.9. а) 2^m; б) 2^n.
1.10. 2) Да. 3) Нет. Невозможно, например, определить смежность вершин 1 и 4, так как пара (1,4) не встречается ни в одном из трех множеств.
1.11. 1) Да. 2) Нет. 3) Нет. 4) Да.
1.12. 1) Да. 2) Нет. 3) Да. 4) Да.
1.13. 1) Да. 2) Нет. 3) Да. 4) Да.
1.14. 1) Да. 2) Да. 3) Нет, графы в левой и правой частях равенства могут иметь разные множества вершин.
1.15. P_4.
1.16. Таких графов не существует.
1.17. 35 вершин, 72 ребра.
1.18. 1) Нет; 2) да; 3) да.
1.19. P_4.
1.20. а) $n(n-1)\ldots(n-k)$; б) $n(n-1)\ldots(n-k+1)$.
1.21. а) $n(n-1)\ldots(n-k)/2$; б) $n(n-1)\ldots(n-k+1)/2k$.
1.22. а) $n(n-1)(n-2)$; б) $n(n-1)(n-2)^2$;
в) $n(n-1)^2(n-2)(n-3)$.
1.23. Не изоморфны.
1.24. Расстояние k, имеется $k!$ кратчайших путей.
1.25. Наибольший диаметр $n-1$, такой диаметр имеет только граф P_n.
1.26. 5 графов.
1.27. Может уменьшиться или остаться прежним.
1.28. 11 графов (это графы вида $K_1 \circ G$, где G – любой граф с 4 вершинами).
1.29. 2 графа.
1.30. rad =diam=3.
1.32. $n(n-1)(n-2)/2$.
1.33. 15 вершин, 44 ребра.
1.34. Любой цикл.
1.35. K_3 и $K_{1,3}$.
1.36. C_3, P_4, K_4, $K_{1,5}$.
1.37. Все, кроме $K_{2,3}$.
1.39. Не может. Если в нем одна вершина степени 50, то в нем одна вершина степени 49. Если эти две вершины смежны в графе, то они несмежны в дополнительном графе и наоборот.

2.1. $2^{\frac{n(n+1)}{2}}$.

2.2. а) 2^{n^2}; б) $2^{n(n-1)}$; в) $3^{\binom{n}{2}}$; г) $2^n 3^{\binom{n}{2}}$; д) $2^{\binom{n}{2}}$.

2.3. $4^{\binom{n}{2}}$.

2.4. $5^{\binom{n}{2}}$.

2.5. а) $2^{\binom{n-k}{2}}$; б) $\sum_{k=0}^{n}(-1)^k \binom{n}{k} 2^{\binom{n-k}{2}}$. Верно.

2.6. $2^{\binom{n-1}{2}}$. Верно.

2.7. а) 1; б) 15.

2.8. $\frac{n(n-3)}{2}$.

2.9. $\left\lfloor \frac{n}{2} \right\rfloor - 1$.

2.10. $\frac{(n-1)(n-2)}{2}$.

2.11. а) 9; б) 12.

2.12. а) $q+1$; б) 60; в) 12.

2.13. а) 12; б) 105.

2.14. а) 2; б) $2n$; в) $p!\,q!$ при $p \neq q$, $2(p!)^2$ при $p = q$; г) 48.

2.15. Наименьший такой граф имеет 6 вершин.

2.16. $f(x) = n + 1 - x$.

2.17. $f(x) = x + j - i \pmod n$ и $f(x) = -x + i + j \pmod n$.

2.19. $\frac{(n-1)(n-2)}{2} + 1$.

3.1. $n - k$.

3.2. n.

3.3. 6 (10) деревьев.

3.4. 82.

3.5. 52.

3.6. 11.

3.9. n^{n-1}.

3.10. 125.

3.11. 3.

3.12. 12.

3.14. В таком дереве 7 вершин.

3.17. 3) и 4).

3.18. а) 3; б) 6; в) 2.

3.19. 12.

3.20. 2^k упорядоченных разбиений.

3.21. При любых.

3.22. $2K_2,\ C_4,\ P_4$.

3.23. $\left\lfloor \frac{n}{2} \right\rfloor \left\lceil \frac{n}{2} \right\rceil$.

3.24. 1) и 3).
3.25. 6.
3.26. 3.
3.27. 2.
3.28. а) 3; б) 3.
3.29. а) 6; б) 194.
3.31. При $k \le 3$.
3.33. $3n - 6$.

4.3. 1), 3), 4).
4.4. Только 4).
4.5 1) 1; 2) 2; 3) k; 4) $\left\lfloor \frac{n}{2} \right\rfloor$; 5) $k + l - 2$.
4.6. Первые два вопроса – «нет», следующие два – «да».
4.7. 1) и 2), для двудольного – только 2).
4.8. а) Могут, достаточно трех переливаний; б) не могут.
4.11. 1), 3), 4).
4.12. Только 4).
4.13. Все.
4.14. 1) $n!$; 2) n; 3) $2n$.
4.15. а) Да (пример – полный граф); б) нет.
4.16. 1) $kl - 1$; 2) $2^k - 1$; 3) $2q$.
4.17. 3) и 4), для двудольного – только 4).
4.19. а) Вершин 23, ребер 33.; б) такого ВС-дерева быть не может; в) вершин 41, ребер 60.

5.1. 4.
5.2. а) 5; б) этот граф нельзя превратить в эйлеров добавлением ребер; в) 4.
5.4. Если стартовать в одной из концевых вершин, результатом будет эйлеров путь, иначе – последовательность вершин, не являющаяся путем.
5.7. 8.
5.8. 1) Есть; 2) нет; 3) нет; 4) есть; 5) есть.
5.9. а) При $p = q$; б) при $|p - q| \le 1$.
5.10. а) При четных n; б) при любых $n \ge 2$.
5.12. 144.
5.15. а) $\binom{n}{3}$; б) $3\binom{n}{4}$; в) $\frac{1}{2k}\frac{n!}{(n-k)!}$.
5.16. а) Да; б) да; в) нет; г) да.
5.17. 12.
5.19. а) $3\binom{n-1}{2}$; б) $4(n-1)^2$.
5.20. а) 2^{n-1}; б) 2^{n-k}.

6.1. а) Таких не существует; б) 4.
6.2. 1) 8; 2) 5; 3)12.
6.3. $\alpha = \beta = 2^{n-1}$, $\omega = 2$.
6.4. а) 2; б) 9; в) 5; г) 1.
6.6. 3^p.
6.7. а) Обе; б) только вторая (пример для первой: в графе P_5 на первом шаге выбирается центральная вершина); в) только первая.
6.9. 1) 9; 2) 4; 3) 5; 4) 18; 5) 20.
6.10. 2 и 6.
6.11. а) Например, C_5; б) не существует.

7.2. 1) 10; 2) 8; 3) 9; 4) 10.
7.3. 6.
7.4. 1) 3; 2) 1; 3) 5; 4) 2; 5) 3; 6) 15; 7) 120.
7.7. 486 чередующихся путей из вершины a, среди них нет увеличивающих, но вообще увеличивающий путь есть.

8.2. 1) 3; 2) 5; 3) 4.
8.5. P_3, C_4, $K_{5,5}$.
8.6. 2) и 3).
8.7. 1) 2; 2) 3; 3) 3; 4) 4; 5) 3; 6) 5.

9.2. 2), 3), 5).

10.3. Все, кроме третьего – если три ребра минимального веса образуют треугольник, то они не могут одновременно входить в каркас.
10.4. 1) $pq - 1$; 2) $(pq - 2)\sqrt{2} + \sqrt{5}$; 3) $63\sqrt{5}$; 4) $120 + 3\sqrt{5}$.

11.2. Второе и третье.

Литература

1. Алексеев В.Е., Захарова Д.В. Теория графов. http://www.unn.ru/books/resources.html. 482.12.08.
2. Алексеев В.Е., Таланов В.А. Графы. Модели вычислений. Алгоритмы.– Н.Новгород, ННГУ, 2005. 307 с.
3. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений.– М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2006. 320 с.
4. Берж К. Теория графов и ее применение.– М.: ИЛ, 1962. 319 с.
5. Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. Лекции по теории графов.– М.: Наука, 1990. 383 с.
6. Зыков А.А. Основы теории графов.– М.: Вузовская книга, 2004. 664 с.
7. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение.– СПб.: БХВ-Петербург, 2003. 1104 с.
8. Кристофидес Н. Теория графов. Алгоритмический подход . – М.: Мир, 1978. 432 с.
9. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ.– М.: Вильямс, 2005. 1296 с.
10. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. 213 с.
11. Оре О. Теория графов.– М.: Наука, 1980. 336 с.
12. Харари Ф. Теория графов.– М.: Мир, 1973. 300 с.

Владимир Евгеньевич **Алексеев**
Дарья Владимировна **Захарова**

ТЕОРИЯ ГРАФОВ

Учебное пособие

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им.Н.И.Лобачевского».
603950, Нижний Новгород, пр.Гагарина, 23

Подписано в печать . Формат 60x84 1/16.
Бумага офсетная. Печать офсетная. Гарнитура Таймс.
Усл.печ.л. . Уч.-изд.л.
Заказ № . Тираж 80 экз.

Отпечатано в типографии Нижегородского университета им. Н.И. Лобачевского
603600, г. Нижний Новгород, ул. Большая Покровская, 37